A Simple and Efficient Bus Management Scheme that Supports Continuous Streams

SAIED HOSSEINI-KHAYAT and ANDREAS D. BOVOPOULOS
Washington University in St. Louis

An efficient bandwidth management and access arbitration scheme for an I/O bus in a multimedia workstation is presented. It assumes that a multimedia workstation consists of a number of processing modules which are interconnected by a packet bus. The scheme is efficient in the sense that it allows the bus to support both continuous media transfers and regular random transactions in such a way that continuous streams can meet their real-time constraints independently of random traffic, and random traffic is not delayed significantly by continuous traffic except when traffic load is very high. Implementation guidelines are provided to show that the scheme is practical. Finally, the performance of this scheme is compared with alternative solutions through simulation.

Categories and Subject Descriptors: C.0 [**Computer Systems Organization**]: General—*system architectures*; B.4.3 [**Input / Output and Data Communications**]: Performance Analysis and Design Aids—*simulation*

General Terms: Algorithms

Additional Key Words and Phrases: Bus arbitration, bus management, continuous stream, multimedia workstation

1. INTRODUCTION

The processing of continuous media (real-time digital video and audio) places special demands on the hardware architecture of computers. One of the main features of a multimedia workstation should be the capability of processing continuous media (CM) as efficiently as discrete data (text, numbers, graphics, etc.). For example, it should be able to receive, transmit, compare, search, transform, store, and display continuous media in real-time. This requires a very high bandwidth interconnect. Although the choice of interconnect is not limited, a bus is a cost-effective interconnect for workstations. This article

This work was supported by the National Science Foundation under grant NCR-9110183 and by an industrial consortium of Ascom Timeplex, Bellcore, BNR, DEC, Goldstar, Italtel SIT, NEC America, NTT, and SynOptics Communications.
Authors' addresses: S. Hosseini-Khayat, Computer and Communications Research Center, Campus Box 1115, Washington University, One Brookings Drive, St. Louis, MO 63130-4899; A. D. Bovopoulos, Chipcom Corporation, 118 Turnpike Road, Southborough, MA 01772-1886.

focuses on the requirements of a bus for a multimedia workstation and proposes an effective and practical solution. The bandwidth of buses in personal computers and workstations has increased significantly in recent years. For example, Sun's XDbus and Intel's PCI bus support a transfer rate of up to 320 and 240MB/sec., respectively. However, increasing the bandwidth is only part of a solution. A multimedia bus must support a mix of random and continuous traffic, and a complete solution should include effective bus bandwidth management to ensure that the real-time constraints of continuous streams are met. Various bandwidth reservation methods have been used in telecommunication systems in general and have been proposed for use in ATM networks in particular [Bae and Suda 1991]. The authors apply the concept to a bus.

The purpose of this article is to introduce a scheduling and management policy that supports a mix of periodic and random jobs and to show how it can be used for a multimedia bus. The bus considered here is a packet-switched I/O bus. We assume that a multimedia workstation is a collection of specialized processing units that cooperate and are linked together by a packet bus (Figure 1).

We believe that a multimedia bus needs to satisfy the following requirements:

(1) It should have a large bandwidth to support continuous and random traffic. As we will show later the required bandwidth depends highly on the bandwidth management method. A well-managed bus requires considerably lower bandwidth than a poorly managed one.
(2) It should provide low latency to random traffic. These transactions arise due to virtual memory page faults, random network transactions, disk transfers, or transmission of data between modules. It is important that they be performed with low latency because they affect the overall responsiveness of a system.
(3) It must maintain a constant throughput for continuous streams in the presence of unpredictable random traffic. We believe that this feature is what a multimedia bus must have to qualify for the name. Ignoring this requirement leads to random momentary freezes of video or frequent loss of video packets and results in poor-quality video.
(4) Its management and arbitration algorithm should be simple so that it can be implemented in fast hardware.

While the fulfillment of the first requirement is currently taking place, design of a bus that meets all of the above conditions is not a trivial problem. This work addresses the last three conditions and proposes an effective solution for a high-bandwidth packet bus.

This article is organized as follows. Section 2 describes a scheduling policy which is the key idea of this article. This policy is adapted for application in a bus in Section 3. In Section 4 we present simulation results and compare the proposed solution with alternative schemes.

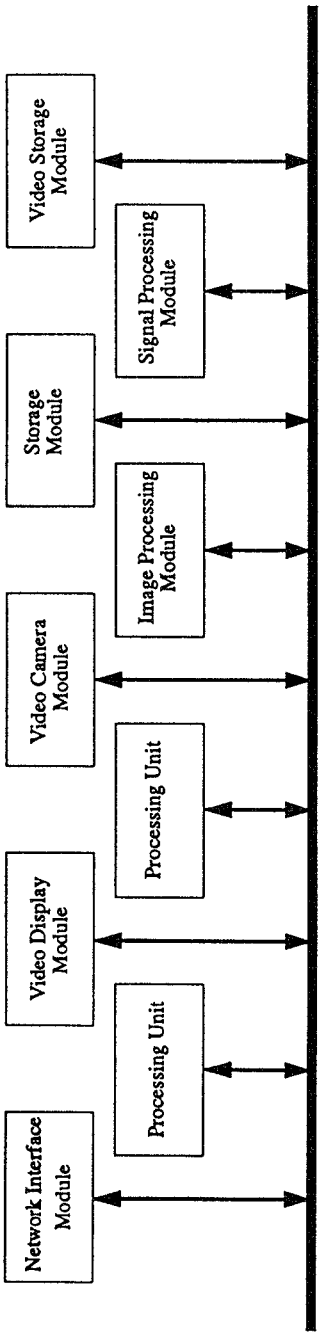

Fig. 1. Example of a multimedia workstation.

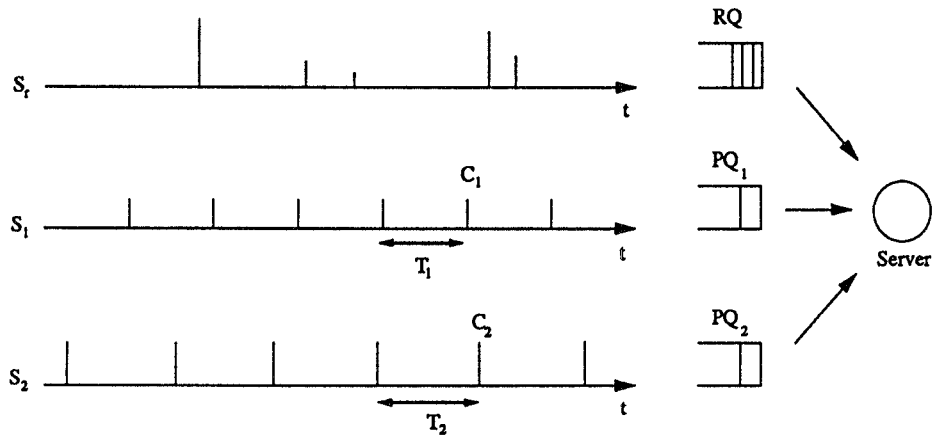

Fig. 2. A server serving two periodic and one random queue.

2. SCHEDULING POLICY

This section describes a scheduling policy that is a key idea in this article. The presentation is general so that it can be used in other applications. In the next section the policy will be modified to fit our application, i.e., a packet bus.

Consider N sources $S_1, S_2, \ldots, S_N$ that periodically generate jobs. Let T_i be the period of source S_i, and let C_i be the service time of each job generated by S_i for all i. Also consider a source S_r that generates jobs at random times $t_0, t_1, t_2 \ldots$ and with corresponding random service times $s_0, s_1, s_2, \ldots$. Suppose that there are N queues $\mathrm{PQ}_1, \mathrm{PQ}_2, \ldots, \mathrm{PQ}_N$ dedicated to periodic sources in corresponding order and that there is a queue RQ for the random jobs (Figure 2). The jobs are placed in queues when they arrive and are to be served in such a way that the following objectives are reached:

(1) The random jobs are served with minimal waiting time. (Waiting time equals time of departure minus time of arrival.)

(2) Each periodic job is served before its source generates the next job. This condition will be referred to as the *backlog avoidance condition*. (We say a backlog happens when a periodic source generates a job before the previous job finishes.)

This is a scheduling problem that needs a rigorous study. In this article we are not looking for its optimal solution(s).[1] Instead we present a practical approach.

[1]It is intuitively plausible that the optimal policy is a modified Earliest-deadline-first policy in which the Earliest-deadline-first (EDF) policy [Liu and Layland 1973] is used to determine priorities among the periodic jobs at any time, and the random jobs delay the EDF schedule so long as there remains enough time for the periodic jobs to finish before their deadlines. However, this seems impractical for a bus, and we do not consider it in this article.

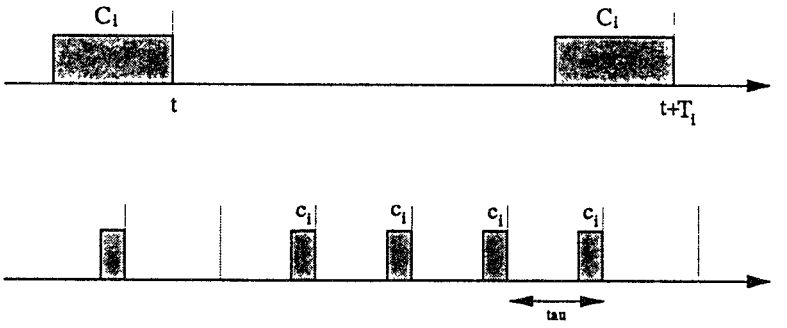

Fig. 3. A periodic job (top) and service cycles (bottom).

In our scheme there is a single server which performs a periodic service routine. Its period is τ and is assumed to be much smaller than any T_i. It does one unit of work per unit time and serves all queues according to the following rules:

(R1) By default RQ has the highest priority. When RQ is empty, the server resumes serving PQs.

(R2) In every period τ a total service time of $\sum_{i=1}^{N} c_i$ is allocated to all periodic sources, and a service time of c_i is allocated to each periodic source, where c_i satisfies $C_i = c_i(\lceil T_i/\tau \rceil - 2)$. The order of service is arbitrary, e.g., in the order of increasing i. The allocated time may be unused but not exceeded.

(R3) RQ loses high priority when the remaining time allocated to the periodic jobs is equal to the remaining time in current period. At this time the server resumes serving periodic jobs.

Rule (R1) tries to fulfill Condition (1), whereas Rules (R2) and (R3) satisfy Condition (2). Rule (R2) guarantees that Source i gets C_i units of service every period T_i. This is true because it gets c_i units of service every period τ and because there are at least $\lceil T_i/\tau \rceil - 2$ periods of τ in a period T_i (Figure 3). This policy breaks the service time of each periodic job into $\lceil T_i/\tau \rceil - 2$ parts and does each part in one service cycle. In addition, it allows the random jobs to preempt the periodic jobs. If preemptions are time consuming, and happen too frequently, the policy becomes inefficient. However, in the case of a packet bus as described later, preemptions do not incur overhead time, and the scheme works efficiently. This policy can also be used for process scheduling as long as the service cycle τ is much larger than the context-switching time. In the next two sections the policy is applied with slight modifications to a packet bus. The next section describes assumptions that this article makes about the architecture, traffic, and operating system of a multimedia workstation.

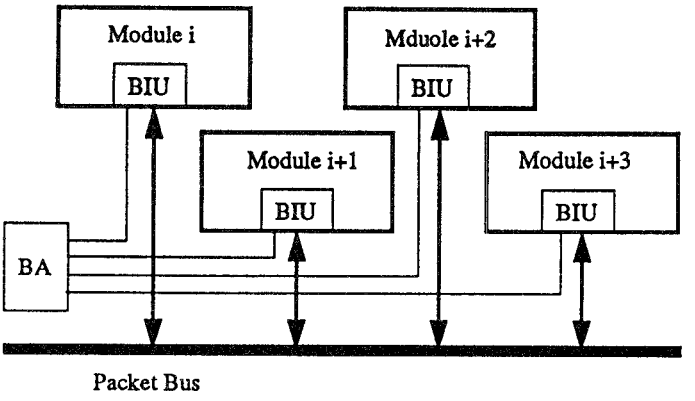

Fig. 4. Bus interface units.

3. ASSUMPTIONS

3.1 Architecture

We assume that a multimedia workstation consists of a number of hardware modules that are connected together by a packet bus. The modules can range from specialized processor boards that include CPU, cache, memory, and bus, to simple peripheral adapters. Every module has a *bus interface unit* (BIU) that connects it to the packet bus (Figure 4). Details of a BIU will be given later. Let us assume that the packets are fixed-sized, and call them *cells*. This assumption is important because it makes preemption and resumption of jobs easier. Interrupting a variable-sized packet and resuming it at a later time is difficult to manage. A cell has a *header*, containing the destination address and other identification information, and a *payload*, containing data (Figure 5). The format and size of cells are immaterial for the scheme. Transmission of a cell is atomic and takes one *time slot* of the bus. In every time slot, the bus master in the next time slot is determined. This is done by a hardware unit called the *bus arbiter* (BA), which is connected to BIUs in every module. (Details of a BA will be explained later.) Therefore, arbitration completely overlaps transmission, allowing back-to-back cell transmission. Preemptions are not expensive in our scheme because they take place only at time slot boundaries.

A BIU contains a cell buffer. The buffer is partitioned (logically or physically) into an input buffer and output buffer, and each partition is divided into random and periodic buffers. They may have variables boundaries that can be adjusted depending on the volume of the continuous or random traffic. One or more of the buffers may not exist depending on the allowable type and direction of traffic. Details about the implementation and management of buffers are not discussed here because the scheme does not depend on them. The scheme only manages the flow of cells from output buffers to input buffers of different modules across the bus. The input buffers do not have any special role in the arbitration of the bus. They hold the incoming cells until

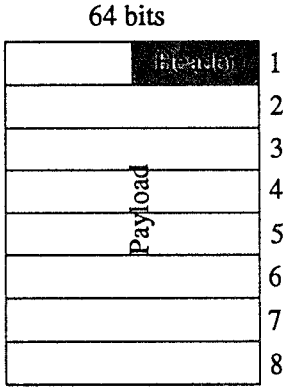

Fig. 5. Example of a cell.

they are consumed by the corresponding module. The size of buffers should also be properly determined based on the volume of expected traffic. If they are too small, and the rate of the incoming or outgoing cells is too high, they may overflow. Issues related to the management and size of buffers are another direction of research which is not pursued in this article. As far as the bus management is concerned, those are details that can be skipped.

Let us focus on the output buffer of a BIU. It has two parts: a *periodic queue* (PQ) and a *random queue* (RQ). A periodic queue holds continuous stream cells, and a random queue holds random cells. Each queue has its own *bus request* (BRQ) and *bus grant* (BGR) signals that are used by the BIU for arbitration. The signal names are indicated in Figure 6. A BIU contends for the bus when either of its output buffers is nonempty. The bus arbiter resolves the contention based on the algorithm that will be explained in Section 4. When a BIU wins, it transmits a cell in the next bus time slot. This process repeats until the queues are empty.

3.2 Traffic

We assume that there are two types of traffic on the bus: random and continuous. The random-traffic type is the usual traffic that exists on buses. It consists of unpredictable bursts of very short duration that convey messages and data between the modules. The continuous traffic, on the other hand, is the result of continuous streams flowing between the modules. A continuous stream (or stream for short) has a very long duration from several seconds to hours and is periodic. It conveys continuous media data between modules that are participating in a multimedia application. As an example scenario, a video camera module sends a continuous stream from a video storage module. The video processor then compresses the resulting video and sends it to a network interface module and a display module. Another continuous stream comes from the network through the network interface module heading for the display module. In the meantime, random cells are transmitted frequently. The continuous streams may have different periods, but the periods are assumed to be much longer than the bus time slots. This is true in practice. For a 64-bit 40MHz bus with a cell size of 64 bytes, a time

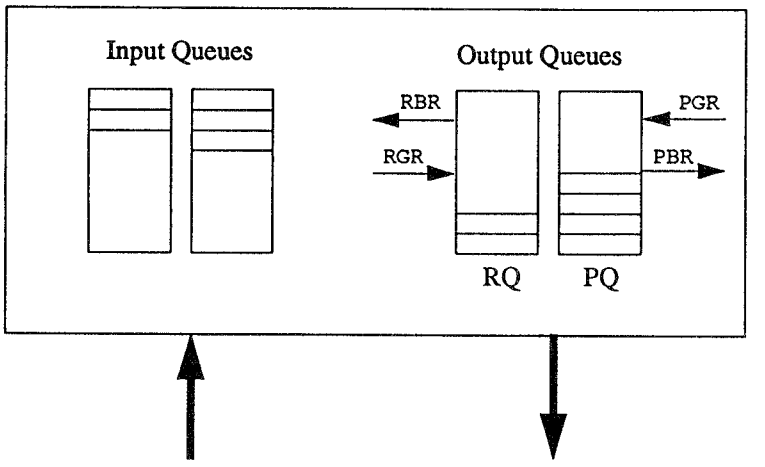

Fig. 6. Queues in a BIU.

slot is 200 nanoseconds, while a video frame time is on the order of milliseconds.

We make no assumption about the statistical characteristics of random traffic. The scheme does not depend on this, and in the absence of continuous traffic the random traffic achieves the same latency that it would without this scheme. It tries to prevent random traffic from being delayed by the continuous traffic. We assume on the other hand that a continuous stream is characterized by two parameters T and C. T is the period of a stream, and C is the number of cells that need to be transferred in every period T. This is what the scheme can guarantee for continuous streams. The parameters T and C completely characterize uncompressed video streams. T is the frame period, and C is the fixed number of cells per frame. For compressed video, the number of cells per frame is variable, and C should be the maximum number of cells in a period T. The resulting unused time slots can be used for random cells. This way of characterization leaves some flexibility to the application. For example, a video module may send 1MB every 1/30 second or 0.5MB every 1/60 second.

3.3 Operating System

The operating system is also required to support the scheme. Its support is in the form of bus bandwidth management. A CM application should pass its continuous stream parameters to the OS and confirm the availability of the required bandwidth. It must reserve bus bandwidth for each of its continuous streams. An operating system daemon, called the *bus manager*, is in charge of taking stream parameters, allocating bandwidth if available, and updating the registers of the bus arbiter. This is done only when a CM application starts or terminates a stream.

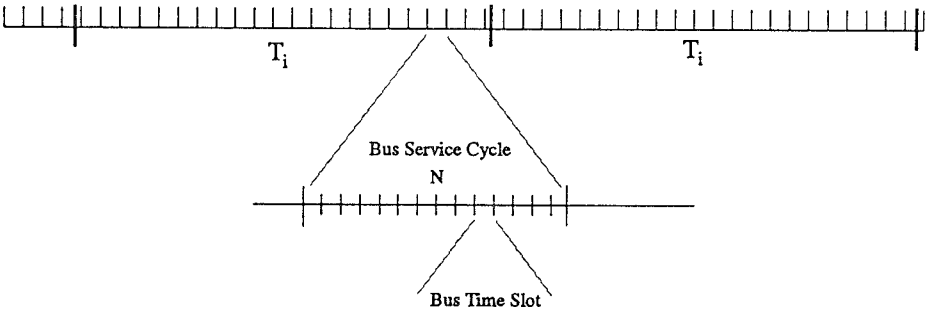

Fig. 7. Source periods, bus service cycles, and bus time slots.

4. PROPOSED SCHEME

4.1 Service Policy

The bus arbiter executes a *service algorithm* periodically with a period of N time slots. N is a system parameter in the range 10–100 that should be set at bootup time. It has the same role as τ in Section 2, but it is discrete. Let us call N the *bus service cycle time* (Figure 7).

Suppose that there are J modules that transmit, and assume for ease of discussion that all of them can source both continuous and random traffic. If there are K_i continuous streams being transmitted from Module i, we assume that they are *paced* by the sending module and placed in the periodic queue of BIU_i. Let C_j^i and T_j^i be the parameters of Stream j in Module i. The pacing rule is:

> For each Stream j, place in PQ_i, M_j^i cells every N times slots, where M_j^i is the smallest integer that satisfies

$$C_j^i \leq M_j^i(\lceil T_j^i/N \rceil - 2). \tag{1}$$

Condition (1) is a discrete version of Rule (R2) in Section 2 and will be explained later. The way pacing is implemented is not important for the bus and is not considered here. As an example, in a processing module pacing can be done by DMA operations from the module's main memory to the BIU buffer.

T_j^i and C_j^i are passed by the application to the bus manager. The bus manager computes M_j^i from (1), applies the admission rule that is presented later, and if bandwidth is available it passes M_j^i to the application and updates variable $Q = \sum_{i=1}^{J} \sum_{j=1}^{K_i} M_j^i$ which is the number of time slots in a bus service cycle that are reserved for continuous streams. Q is updated every time a stream is admitted or terminated. After Q is updated, its value is written into a special register in the bus arbiter. Figure 8 shows the flow of parameters among the bus manager, the bus arbiter, and a continuous media application.

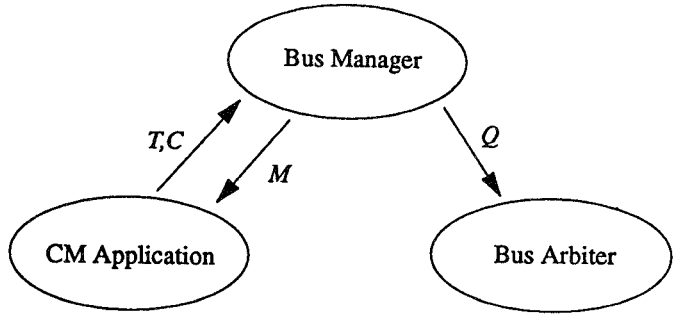

Fig. 8. Flow of parameters.

All CM cells outgoing from a module are placed in the periodic queue of the module's BIU. Therefore, there are $\Sigma_{J=1}^{K_i} M_J^i$ CM cells that are supposed to go out from Module i in every N time slots. Comparing this to the scheme in Section 2, here there are J queues $\mathrm{PQ}_1, \mathrm{PQ}_2, \ldots, \mathrm{PQ}_J$ that hold periodic jobs generated by J sources (i.e., Modules $1, \ldots, J$).

The outgoing random cells in a module, on the other hand, are placed in the random queue of that module. Therefore, there are J random queues that collectively have the role of RQ in Section 2. Each cell in a random queue is a random job that must be served by the bus.

PQs and RQs contend for the bus when they have cells. The bus arbiter is responsible for resolving the contention. It has two hardware registers N and Q which hold the value of N (the bus service cycle time) and Q (the number of reserved time slots for continuous streams) and has two counters n and q that are reloaded every service cycle from N and Q, respectively. In the following, n is the value of Counter n and denotes the remaining number of time slots in a service cycle (initially N), and q is the value of Counter q and denotes the number of time slots allocated to CM cells that have not been used by CM cells yet during this service cycle (initially Q). The bus arbiter does its job in every time slot according to the following rules:

(r1) If $n > q$, grant the bus to an RQ if any of them is requesting; otherwise grant the bus to any PQ if any of them is requesting.

(r2) If $n \leq q$, grant the bus to a PQ if any of them is requesting; otherwise grant the bus to an RQ if any of them is requesting.

(r3) Decrement q if grant is given to a PQ.

(r4) Decrement n. If $n = 0$, reload counter n from register N and counter q from register Q.

This set of rules is a practical version of Rules (R1), (R2), and (R3). Rule (r1) gives priority to random cells, when in a service cycle, the remaining time n is more than q. Rule (r2) gives priority to CM cells when the remaining time n is less than or equal to q. Therefore, a service cycle is divided into two intervals. (This is assuming $Q \leq N$, a condition that is ensured with the

admission rule that is explained later.) The first interval starts with the start of a service cycle and ends when $n = q$ is reached. During this interval, random traffic has priority. Any time slot unused by random traffic can be used by a CM cell. If this happens, both n and q are decremented. Otherwise only n is decremented. The second interval starts when $q = n$ and ends when a service cycle ends. In this interval continuous traffic gets priority over random traffic. It is interesting to note that the boundary between the two intervals moves gradually toward the end of the service cycle when CM cells are served in the first interval. With moderate random and continuous traffic load there is a good chance that in the first interval (1) some time slots are not used by the random traffic and (2) some CM cells are served. When this happens, q decreases; and as a result the first interval is stretched, and the second interval (where random cells get low priority) shrinks. This implies that random traffic tends to remain in high priority most of the time and does not notice the continuous traffic. In our simulation results, this effect is clearly shown. The mean delay of random cells in the presence of continuous traffic can approach the value of mean delay in the absence of continuous traffic.

The above set of rules does not specify any priority among RQs or among PQs and leaves this freedom to the designer. It also assumes that the modules are cooperative and do not exceed the limits assigned by the bus manager.

4.2 Admission Rule

The admission or rejection of new continuous streams is done by the bus manager. A request for a new continuous stream is sent to the bus manager. Let us call its parameters C_{new} and T_{new}. The bus manager computes M_{new} from (1) and admits the stream if the following condition holds:

$$M_{new} + Q \leq N - \alpha, \tag{2}$$

where Q is the number of time slots in a cycle reserved for continuous streams, and α denotes the number of time slots dedicated for random traffic. α is a system parameter that should be changed only when a change in the fraction of allowable continuous traffic is desired. $1 - \alpha/N$ is the maximum fraction of bandwidth that can be allocated to continuous traffic. Increasing α increases the responsiveness of the system by devoting more bandwidth to random traffic.

If the stream is admitted, Q is incremented by M_{new}. Termination of a stream is signalled to the bus manager by the application. The bus manager decrements Q by parameter M of the corresponding stream. After any change in Q, its value is written in Register Q in the bus arbiter.

4.3 Backlog Avoidance

The scheme guarantees backlog avoidance for continuous streams as defined in Section 2. This is achieved by Rule (r2) in conjunction with the pacing rule. As explained earlier each stream is paced and puts in every service cycle up

to a certain number of cells in a periodic buffer. This prevents it from flooding the bus. Rule (r2) guarantees that at least those cells are served in a service cycle.

Let us focus on a single stream with parameters T and C. In order to ensure that C cells are served every T time slots, the bus must allocate M times slots to that stream in every N time slots. Condition (1) which determines M is based on the fact there are at least $\lceil T/N \rceil - 2$ complete service cycles in an interval of length T. The service cycles at the two ends of the interval may be partially covered. Since it is not certain where in a service cycle a CM cell is served, the partially covered cycles must be excluded. Allocating M time slots in each of the completely covered service cycles guarantees that at least $M(\lceil T/N \rceil - 2)$ cells of the stream can be served in every period T. This number is larger than C according to Condition (1). Therefore, a backlog never occurs for the stream in question. The same argument holds for every stream, and a total of $Q = \Sigma M$ is allocated to all streams.

Obviously Condition (1) slightly overallocates. There may be service cycles with a number of CM cells less than the allocated number. For example, when $T = 17500$, $C = 1650$, and $N = 85$, Condition (1) gives $M = 9$. In terms of the percentage of the bus bandwidth, the stream requires 9.4% (obtained from C/T) while Condition (1) allocates 10.6% (obtained from M/N). There are at least 204 complete service cycles within one period. Therefore the stream has 1836 ($= 204 \times 9$) time slots to use in one period, but it only generates 1650 cells in one period. Although overallocation exists, it does not take away time slots from the random traffic. Rule (r2) allows the random cells to use the bus when there are no CM cells in the allocated time interval, i.e., the interval where $n \leq q$.

4.4 Implementation

The service policy in Section 4.1 has a simple realization. Every module has a BIU which connects it to the bus and has two output queues: RQ and PQ. Each queue has its own bus request and bus grant signals and contends for the bus independently. Figure 9 schematically shows RQs and PQs that are tied to the bus. Other details, including the input buffers, are left out because they do not have any role in arbitration. The bus arbiter consists of two registers, Q and N, and two counters, n and q, and some conbinational logic circuits. At any time slot if PQ_i (RQ_i) has a cell it asserts a bus request signal $PBRQ_i$ ($RBRQ_i$). The bus arbiter asserts or de-asserts a bus grant signal $PBGR_i$ ($RBGR_i$) depending on the value of counters n and q.

Two types of implementation are discussed here: serial and parallel. Hybrid implementations are also possible. In a parallel design, all bus request signals are input to a combinational circuit in the bus arbiter which works in every time slot based on the following logic:

(1) Compare Counters n and q. If $n > q$ and any of RBRQ signals are asserted, then assert RBGR to RQ with the highest priority. Otherwise if any of PRBQ signals are asserted, then assert PBGR to PQ with the highest priority. If no bus request signal is asserted, then assert no PBGR.

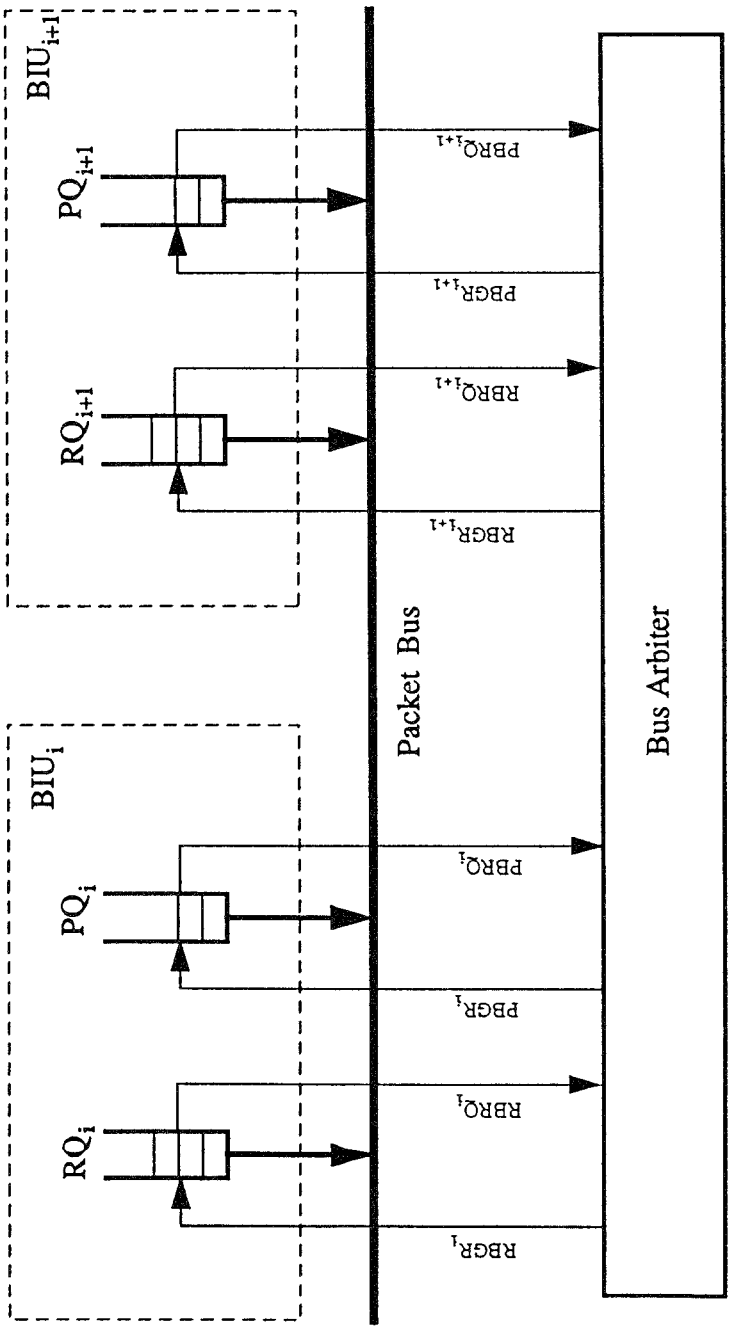

Fig. 9. Parallel arbitration.

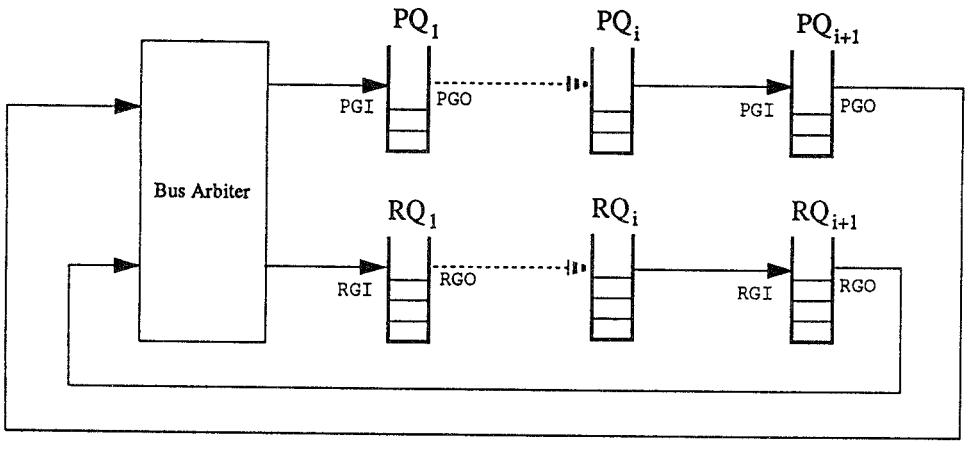

Fig. 10. Serial arbitration.

If $n \leq q$ and any of PBRQ signals are asserted, then assert PBGR to PQ with the highest priority. Otherwise if any of RBRQ signals are asserted, then assert PBGR to RQ with the highest priority. If no bus request signal is asserted, then assert no PBGR.

(2) Decrement q if a PBGR is asserted.

(3) Decrement n. If $n = 0$, reload counter n from register N, and counter q from register Q.

Note that priorities among RQs (PQs) are not specified in the above. When the circuit is designed, these priorities can be assigned to signals RBRQ_i (PBRQ_i) in the increasing order of index i, and when the system is designed, those signals can be tied to the BIUs with corresponding priorities.

In a serial design, shown in Figure 10, there are two closed daisy chains. All RQs (PQs) and the bus arbiter form a closed daisy chain. The priority order among RQs (PQs) is implicit in the arrangement of the chain. The queue immediately after (in the directed loop) to the bus arbiter gets the highest priority, and the one immediately before (in the directed loop) the bus arbiter gets the lowest priority.

In a closed daisy chain, a bus grant signal is generated by the bus arbiter, and it passes through every queue in the chain and finally returns to the bus arbiter. Any queue in a chain can block a grant, if it gets one, and become bus master.

Every RQ (PQ) has an input signal RGI (PGI) and an output signal RGO (PGO). These have the following relation with bus request and bus grant signals:

$$\text{RGO} = \text{RGI} \vee \overline{\text{RBRQ}}$$
$$\text{RBGR} = \text{RGI} \vee \text{RBRQ}$$
$$\text{PGO} = \text{PGI} \vee \overline{\text{PBRQ}}$$
$$\text{PBGR} = \text{PGI} \vee \text{PBRQ}$$

where $\vee$ is the logical OR, and signals are assumed "asserted low."

```
if n > q then
   assert RGO;
   wait t_p;
   if RGI = low then
      assert PGO;
      wait t_p;
      if PGI = high then
         q ← q − 1;
      fi
   fi
else
   assert PGO;
   wait t_p;
   if PGI = high then
      q ← q − 1;
   else
      assert RGO;
   fi
fi
n ← n − 1;
```

Fig. 11. Serial arbitration algorithm.

An RQ (PQ) blocks an RGI (PGI) signal if it has a cell. Let us assume that a grant signal from the bus arbiter takes t_p to propagate around the loop when no queue blocks it. In every time slot, the bus arbiter performs the algorithm in Figure 11.

Both serial and parallel implementations presented here are simple and fast. When a chain propagation delay is larger than a time slot, the serial design is not practical. In that case, a parallel implementation should be adopted.

5. SIMULATION AND COMPARISON

We simulated and compared our scheme with two alternative methods.

In the first alternative method, which we call *method A*, in every module the continuous streams are paced according to the pacing rule (1) in Section 4.1 and are placed in a single output queue that is shared with the random cells. There are no dedicated queues for random or continuous traffic, and there is one bus request and one bus grant signal associated with each queue. The output queues in each BIU then contend for the bus. A conventional bus arbitration method (e.g., a single daisy chain, or a distributed contention resolution circuit [Ward and Halstead 1990]) is used which resolves the contention based on a fixed priority ranking of the modules. The cells in a queue are served in FIFO order, so there is no priority between random and continuous cells.

The second method, which we call *method B*, uses a different pacing rule for continuous streams, and uses dedicated output queues (RQs and PQs as in the proposed scheme) for each type of traffic. The pacing rule is: "Place 1 cell of Stream i with parameters C_i and T_i in a periodic queue every N_i time

slots where N_i is the largest integer that satisfies $C_i/T_i \leq 1/N_i$." Here there is no common fixed service cycle, and N_i can be different for each stream. A conventional bus arbiter resolves the contention by giving PQs priority over RQs. Therefore CM cells *always* have priority over random cells.

Method A represents the idea that no special treatment is needed for continuous streams, and "sufficient" bandwidth along with pacing solves the problem. Method B, on the other hand, represents the idea that continuous streams must always be given priority over random traffic.

In our simulations there are five modules capable of sending and receiving continuous and random cells. The continuous traffic consists of five streams with parameters (T, C) as follows: (2520, 63) from Module 1, (2520, 63) from Module 2, (2520, 63) from Module 3, (2520, 126) from Module 4, and (1512, 567) from Module 5. (We did a number of experiments with different parameters. Those results were consistent with the results obtained from the above parameters that are reported here.) Note that the destinations need not be given, because a bus is a broadcast medium. Those figures were calculated such that they roughly represent a bus with 320MB/sec. bandwidth carrying four compressed and one uncompressed video in the form of 64-byte cells. The random traffic was simulated by arrival of cells at RQs according to a Bernoulli trial model. A cell was placed in an RQ at any time slot with probability $p/5$. The total random load p was changed in different runs to get Figures 12 and 13. A Bernoulli model does not fit real random traffic, because it lacks burstiness and correlation that exists in practice, but it should give us an indication of performance of each method.

In our comparison, we were interested in the mean delay of random cells. The requirement for the continuous streams (backlog avoidance) was met in our simulations of the proposed scheme. No CM cells missed their deadlines. Method A does not guarantee deadlines for CM cells, because they are served in FIFO order with random cells. Method B performs better in this aspect, because it always gives priority to CM cells. But it does not strictly guarantee deadlines. The pacing rule in Method B only spreads CM cells in time, but it does not guarantee that they get through within a specified time. Occasionally some CM cells may miss their deadlines because of contention with other CM cells. It also results in higher mean delay of random cells.

The mean delay of random cells in the proposed scheme is significantly less than the two alternative methods. Figure 12 shows mean delay versus total random load (p). The total continuous load $\Sigma C/T$ and the bus service cycle time N were kept fixed at 0.5 and 40, respectively. It is seen that at $p = 0.3$ (a total load of 0.8), our mean delay is very close to 1, while mean delay for Methods A and B is close to 5. The mean delay at $p = 0.4$ (a total load of 0.9) is well above 20 for the alternative methods, while the proposed scheme gives a mean delay of 1.5. The plot also shows that a mean delay equal to about 15 happens for Methods A and B at a total load of about 0.86 ($p = 0.36$), while the same delay happens for the proposed scheme at a total load of 0.98 ($p = 0.48$). This shows that efficient bus management is much better than over-engineering (which is the idea behind Method A). Note that the mean delay for Method A is always less than mean delay for Method B. The reason

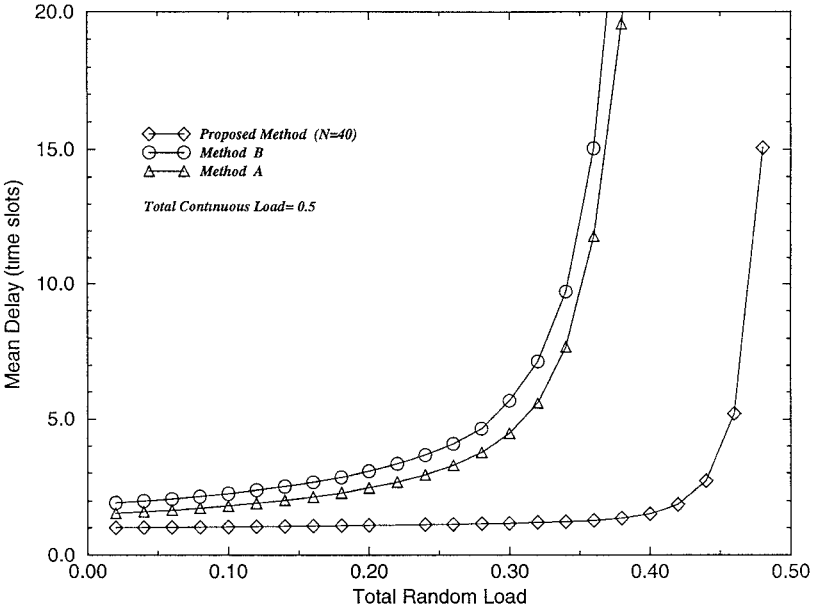

Fig. 12. Mean delay.

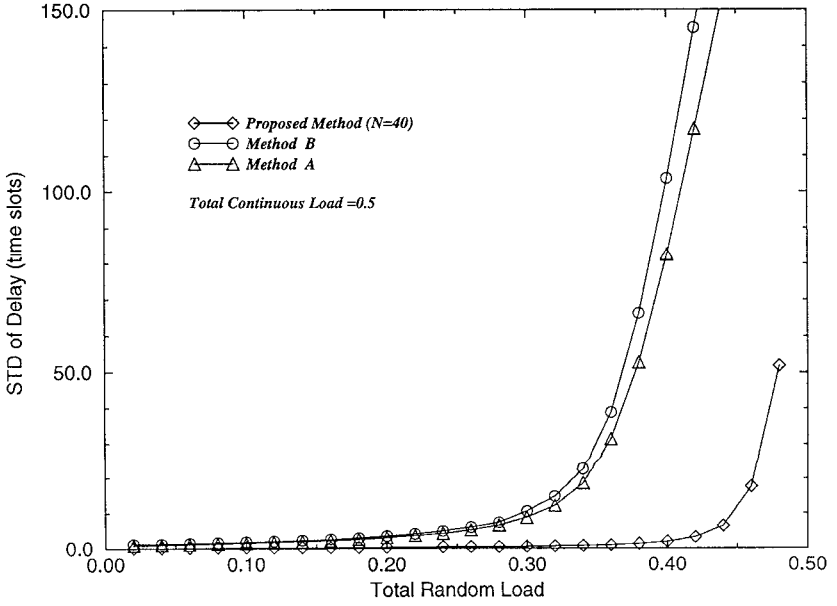

Fig. 13. Standard deviation of delay.

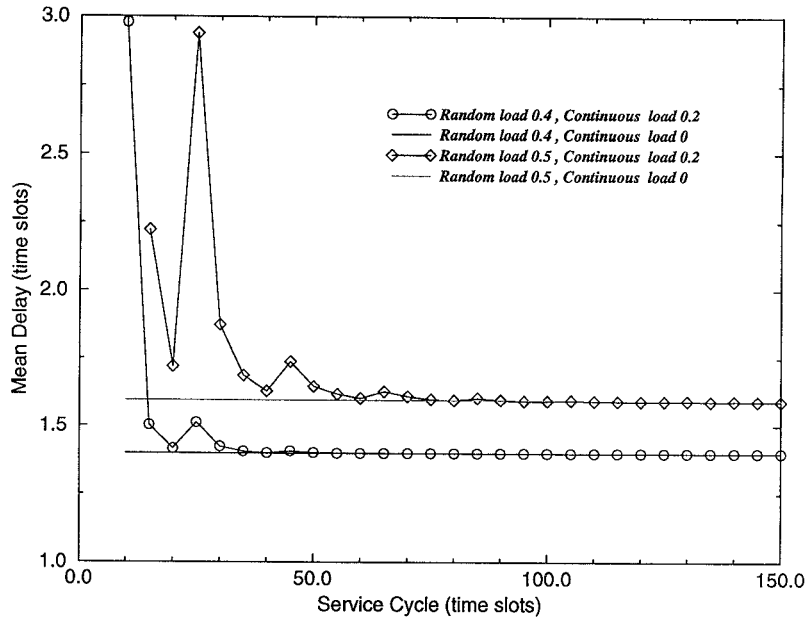

Fig. 14. Mean delay vs. N.

should be obvious. Method B always gives priority to CM cells while Method A serves in FIFO order.

Figure 13 shows the standard deviation of delay of random cells in the same simulation. It shows that uncertainty of delay is also low for the proposed method.

We did another experiment which is shown in Figure 14. It shows mean delay versus service cycle time N for two sets of load values. First, the continuous streams were shut down, and the delay of random cells was measured. At total random load of 0.4 and 0.5 the mean delay is 1.36 and 1.6, respectively. These give the two horizontal lines in Figure 14. (In the absence of continuous traffic, the size of a service cycle has no effect on the mean delay of random cells. Therefore mean delay is constant.) Then, the continuous streams were turned on. The total continuous load was kept constant at 0.2, while in each run N was increased starting from 10 in steps of 5. The mean delay of random cells was measured. The same procedure was done for two values of total random load: 0.4 and 0.5. The plot is interesting because it shows that at low N the difference between the two cases (with and without continuous traffic) is quite visible. As N increases, mean delay decreases until it becomes very close to the horizontal line (i.e., the mean delay in the absence of continuous traffic). This effect, which can be called the *hiding of continuous traffic*, is due to the fact that the random cells notice fewer and fewer CM cells as the service cycle of the bus becomes longer. This is possible because the proposed scheme takes advantage of the trade-off between the delay of random cells (which should be low) and the delay of CM cells (which

can be made large up to some point). The same effect was seen with different values of traffic parameters as long as the total load was not close to 1.

6. CONCLUSION

This article discussed the requirements of a multimedia I/O bus and proposed a bandwidth management and arbitration method that guarantees the timing requirement of continuous streams and does not cause excessive delay of random traffic. We showed that over-engineering is not a good solution, because it needs significantly larger bandwidth to provide performance comparable to a well-managed bus. The scheduling algorithm presented has the potential of solving a similar problem in process scheduling.

ACKNOWLEDGMENT

The authors appreciate the help of their colleagues in the Computer and Communications Research Center. Helpful comments from Jonathan S. Turner at early stages of this work increased its practical flavor. Discussions with Hossein Saidi and Raman Gopalakrishnan resulted in more realistic refinements. Finally, J. Andrew Fingerhut added to its clarity by his careful reading and suggestions.

REFERENCES

BAE, J. J. AND SUDA, T. 1991. Survey of traffic control schemes and protocols in ATM networks. *Proc. IEEE 79*, 2 (Feb.)

DI GIACOMO, J. 1990. *Digital Bus Handbook*. McGraw-Hill, New York.

LIU, C. L. AND LAYLAND, J. W. 1973. Scheduling algorithms for multiprogramming in a hard-real-time environment. *J. ACM 20*, 1 (Jan.).

LYLE, J. D. 1992. *SBus, Information, Applications and Experience*. Springer-Verlag, New York.

PATTERSON, D. A. AND HENNESSY, J. L. 1990. *Computer Architecture, A Quantitative Approach*. Morgan Kaufmann, San Mateo, Calif

SUN. 1992. SPARCcenter 2000, architecture and implementation. Tech. White Paper, Sun Microsystems, Inc., Palo Alto, Calif. Nov.

WARD, S. A. AND HALSTEAD, R. H. 1990. *Computation Structures*. The MIT Press, Cambridge, Mass.