

# Scheduling Algorithms for A High-Speed Switch Supporting Real-Time Periodic Traffic Sources

Jonathan C.L. Liu, Lin Xia  
CISE Dept.  
University of Florida

David H.C. Du  
CSE Dept.  
University of Minnesota

Rose Tsang  
Sandia National Laboratory

A. P. Aarn  
Honeywell Technology Center

## Abstract

The successful operation of mission critical systems requires a sophisticated control network which provides for the real-time delivery of data from a very large number of diverse sources such as sensors, audio/video surveillance, computational sources, etc., as well as remote monitoring and diagnosis sources. It is expected that many of the traffic sources will be in the form of periodic sources with real-time requirements. In this study, we propose fast scheduling algorithms for multiplexing periodic source flows with real-time delivery requirements. We assume a single switch environment with periodic source flows being supported over a Constant Bit Rate (CBR) type circuit. We undertake a systematic study beginning from the simplest scenario where all traffic flows consist of data frames of the same size and with the same real-time delay requirement. In this case, we prove that the FCFS algorithm is optimal. We conclude with the most difficult case of examining CBR flows of variable bandwidth requirements and variable delay requirements. Algorithms and analysis are presented for all the cases. The simulation results show the substantial performance gains provided by the proposed algorithms.

## 1 Introduction

Support for isochronous traffic streams will be an important service in future networks. A variety of multimedia applications are already pushing the limits of current networks primarily in their requirements for high bandwidths and support for guaranteed real-time delivery. Furthermore, computer networks are increasingly finding use in today's control systems controlling a complex arrangement of sensors and actuators in highly varied applications such as building climate control, flight navigation, factory automation, and mission command and control in military uses [7]. The most common element

in these control systems is the presence of numerous sensors which typically generate short bursts of information in a highly periodic fashion (e.g. a sensor monitoring a coating process will draw light samples from the coating at periodic intervals).

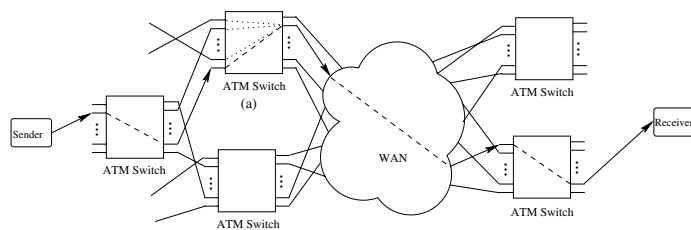


Figure 1: A WAN Environment with Multiple Switches

With the rapid development of these applications, network support for real-time periodic source traffic flows is becoming increasingly important. In a typical wide area network (WAN), as shown in Figure 1, a data unit (packet) will usually pass multiple switches on its route from the sender to the receiver. In order to support the real-time requirements of these periodic sources, switches must provide guaranteed delay through a specialized scheduler. Our study focuses on scheduling algorithms for a single switch.

In any network, the total delay for one data unit consists of three factors: transmission delay, propagation delay, and processing delay in the switches on route to the destination. The processing delay at the switches in turn consists of switching delay (time needed to look up routing tables, and do header translation) and the delay incurred in waiting in buffers at input and/or output ports. For a given virtual circuit, the transmission delay and propagation delay are constant. The only variable factor is the delay incurred in the buffers. In most high-speed switching products, output buffering and/or buffer sharing is used to achieve maximal throughput and minimal buffering space [1, 8, 11]. Our study assumes output buffering, which is the current industry trend in ATM switch design. However, the algorithms

\*This work is supported in part by Advanced Research Projects Agency (DARPA) through AFB, contract number F19628-94-C-0044, National Science Foundation under Grants CCR-9874726, CDA-9502979, CD A-9414015 and CIA-9422044. Please address all the correspondences to jcliu@cise.ufl.edu.

presented in this paper may be generalizable to other switch architectures.

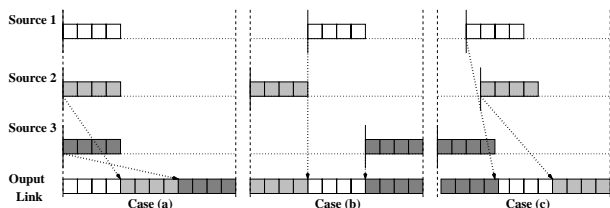


Figure 2: Three Cases of Total Delay from Different Arrival Patterns

As an example of cases where contention for the same output port generates variable delay is depicted in Figure 2. These variable delays can have a wide range of values depending on how the incoming Virtual Circuits (VCs) are treated by the switch scheduler. For example, in Case (a) of Figure 2, all three sources arrive at the same time simultaneously contending for the same output link. Since the output link can only serve one source at one time, the other two sources need to be buffered, i.e., delayed. One source needs to be delayed a time period of 4 data units and the other source needs to be delayed a time period of 8 data units. Therefore, the total delay is 12 data units. By adopting a FCFS (First Come First Serve) scheduling policy, the total delay in Case (b) is zero because all sources are served by the output link upon their arrivals. Case (c) depicts a random case in which the delay for the individual sources are 0 units for Source 3, 2 units for Source 1, and 5 units for Source 2. Therefore, the total delay in this case is 7 data units.

Different VCs may have different sizes of periodic data units and different periodicities. For all these different cases, the traditionally used FCFS policy might not result in the necessary real-time performance. An exhaustive search of all possible solutions will find an optimal schedule, however, the fast switching necessary at these switches precludes the use of computationally complex scheduling algorithms. This paper presents a study of scheduling algorithms for real-time periodic traffic sources to be used in high-speed network switches (e.g., ATM switches). The goal is to develop fast algorithms that generate feasible schedules, which provide guarantees to meet the real-time requirements of the application traffic.

We adopt a systematic approach beginning from the case where all competing VCs have a common deadline. This case can be further decomposed into two sub-cases: (i) all competing VCs are carrying messages with the same frame size<sup>1</sup>, and (ii) all competing VCs may impose different frame sizes. The FCFS algorithm is an-

<sup>1</sup>A *frame* in this paper is defined as a message delivered in the network with a back-to-back manner.

alyzed and proved to be optimal for the case where all competing VCs carry the similar-sized frames. When the frame sizes are different among different VCs, special algorithms are needed to improve the performance over the FCFS algorithm. We propose a heuristic algorithm to schedule the competing VCs on the fly when they are carrying frames with different sizes. Our experimental results show that when frames of different VCs arrive at the same time as shown in case (a) of Figure 2, the reduction in processing delay achieved by our proposed algorithm range between 38% and 75% compared with the traditional FCFS algorithm. When assuming random arrival pattern, as shown in case (c) of Figure 2, the reduction is 6% when there are 8 competing VCs and increases to 38% with 32 competing VCs. Compared to the  $O(N!)$  time complexity required for exhaustive search, the time complexity for our proposed algorithm is  $O(N)$ .

We furthered our investigation to examine cases where different VCs had different deadline requirements for their frames. Here we differentiate the concept of *deadline* from the commonly defined *period*. Real-time control applications such as military  $C^4I$  systems may impose a strict deadline for the completion of processing of each message within an ATM switch. For example, a sensor may generate a periodic data stream of 30 frames per second which corresponds to a period of 33 msec. However, the deadline for an ATM switch to finish the processing of each frame may be shorter than 3 msec if the message is to trigger a crucial alarm system. We thus classify traffic with different deadlines in two sub-cases: (i) all competing VCs are carrying the messages with the same frame size and (ii) all competing VCs may contain different frame sizes. We propose a heuristic algorithm which incorporates a form of classical EDF (Earliest Deadline First) scheduling algorithm with special consideration of the periodic traffic sources. The experimental results for this case show that when the number of competing VCs is small, our proposed algorithm can meet the deadline requirements of all the frames while the traditional FCFS algorithm may cause some frames to miss their deadlines. When the number of competing VCs is increased to 32, our proposed algorithm can reduce the number of frames missing deadline by 88% compared to the FCFS algorithm. The time complexity of this algorithm is also  $O(N)$ .

The case of different frame sizes among the competing VCs introduces the most challenging problem because the varying frame sizes and deadlines need to be jointly considered. The problem of guaranteeing both goals is a very difficult task. To the best of our knowledge, guaranteeing both goals at the same time is indeed an open area and there are no clear solutions. Thus we chose to, first, meet the deadline requirement, and second, based upon the first goal, minimize the total processing delay. A heuristic algorithm satisfying the above criteria is pre-

sented. The experimental results illustrate that (1) the proposed algorithm achieves significant performance improvements over the traditional FCFS algorithm. When the frames of competing VCs arrive in a random pattern (as in part 'c' of Figure 2), the reductions in processing delay increase gradually from 1.5% for 2 competing VCs to 36.0% for 32 competing VCs. The proposed algorithm also reduces the number of frames missing deadlines by a range of 91% and 99% compared with the FCFS algorithm; (2) The proposed algorithm for the general case outperforms the other algorithms in this paper on balancing the two performance goals. The time complexity of this algorithm is  $O(N^2)$ .

## Related Work

Various scheduling methods have been proposed in recent years. VirtualClock [5, 17] assigns a *time stamp* to each cell and uses this timing information to decide the scheduling priority. A similar approach is proposed in [12] using a global clock generated at the physical layer. Other schedulers using timing information include Delay-EDD (Earliest-Due-Date) [6] and PGPS (Packet-Generalized-Processor-Sharing), which is also called WFQ (Weighted-Fair-Queuing) [14, 6]. Particularly, [10] presented a solid analysis and simulation study on multiplexing video conferencing traffic. Nevertheless this work only addressed the cell loss probability, but did not address our concerns on minimizing the processing delay with deadline preservation.

There are also related literatures from real-time computing domain. [15] reviewed the classical scheduling algorithms including Rate Monotonic Algorithm. Our proposed algorithms are particularly designed for communication aspects, thus the pseudo-code algorithms can be more practical to the system designers in the communication field. Since our proposed algorithms focus on data-frame-level scheduling, it is very difficult to compare the above algorithms with the algorithms proposed in our paper. Moreover, our analysis considers both the timing information (deadline) and the data rate (frame size) jointly using a systematic approach, which the previous studies did not address the same goals in a similar and complete manner.

The remainder of this paper is organized as follows. We describe the problem in Section 2. Scheduling algorithms to support competing VCs with a common deadline are introduced in Section 3. Section 4 discusses the heuristic scheduling algorithms to support competing VCs with different deadlines. Empirical results using these heuristic algorithms are presented in Section 5. We present our conclusions in Section 6.

## 2 Problem Description and Formulation

In this section, we describe the problem and formulate it mathematically. As mentioned before, the environment we assume is an ATM LAN/MAN network, as illustrated in Figure 1, which consists of multiple ATM switches. The edge switches are fed traffic from multiple sources where each source is generating a data stream, as described in the previous section.

We formally define each individual CBR stream as an isochronous traffic source,  $VC_i$ , which consists of periodic segments of data, called *frames*, where each frame is of constant size,  $s_i$ . The unit of size will be assumed to be in terms of *cells*. The delay requirement for each frame at each switch is *deadline<sub>i</sub>*. The unit of time used will be denoted as a *cell time* or *slot*. A *cell time* or *slot* is the time it takes to transmit one cell.

The following assumptions are used in this study.

- We assume a generic output-buffered switch providing full cross-connectivity between all input and output ports. All the frames competing for the same output link are buffered in one single queue of that output link. However, the frames from the same VC are linked in a *sub-buffer-queue* for that VC. This can be implemented by organizing the frames from different VCs in different logical queues within a single physical queue, as illustrated in [4, 9]. Therefore, if there are  $N$  VCs competing for the same output link, there will be  $N$  *sub-buffer-queues* inside the logical buffer queue of that output link.
- Individual source flows are all considered to be of equal priority and all of them are isochronous streams of the type described above.
- Appropriate call-level admission control is performed at the edges of the network so that for any interior switch the total input bandwidths cannot exceed the output link capacity bandwidth.
- All scheduling algorithms we consider are *work conserving*, i.e., if there exists an available cell to be scheduled and the output link is idle, it must immediately be scheduled on the idle output link.

**Definition 1** *The delay experienced by a frame within a switch, denoted by  $\delta_i$ , is the time difference between the time its first cell is transmitted out of the switch and the time the first cell arrives at the switch.*

As discussed in Section 1, the total experienced delay are different for various arrival patterns and scheduling policies. We formulate the general problem as follows.

*Given a switch with  $N$  incoming competing VCs. Each  $VC_i = (s_i, \text{deadline}_i)$  consists of the periodic delivery of frames of size,  $s_i$ , where each frame has a deadline*

requirement,  $deadline_i$ . Find a service scheduling policy among these  $N$  competing VCs that minimizes the sum of the delay of all the VCs, i.e., the total delay,  $\Delta_{min}$ .

$$\Delta_{min} = \min\left(\sum_{i=1}^N \delta_i\right), \quad (1)$$

while maximizing the number of VCs which meet their deadlines.

It is certainly true that an optimal schedule can be found by examining all the combinations which consists of  $N! = N * (N-1) * (N-2) * \dots * 2 * 1$  possibilities. Obviously, the high-speed processing requirement of an ATM switch would prohibit such an expensive computation. In the following sections we propose heuristic algorithms as solutions to the above defined problem.

### 3 Scheduling Algorithms for A Common Deadline

In this section, we assume the case where frames from all the incoming VCs have the same deadline, i.e.,  $deadline_i = deadline_j$ , for all  $i, j \leq N$ . We examine two cases of the common deadline. In the first case, all the frame sizes of all the VCs are the same,  $s_i = s_j$  for all  $i, j \leq N$ . In the second case frame sizes may vary among VCs, i.e., different size frames will be contending for the output link.

#### 3.1 With Same Frame Size

When the frame sizes are the same among competing VCs, we examine the simple FCFS scheduling policy.

**Lemma 1** Assuming all VCs have the same frame size,  $s_i$ , and all frames have the same common deadline,  $deadline_i$ , the service order does not have an effect on the total delay,  $\Delta$ . Thus the FCFS policy with the first incoming VC as the reference will produce the minimal total delay.

**Proof.** Let  $current_i$  denote the number of cells in the current frame of  $VC_i$  which have already arrived (are currently buffered). Let  $remain_i = s_i - current_i$ , and let  $\Delta_i$  be the total delay if the frame from  $VC_i$  is scheduled first.

If the scheduler chooses to schedule  $VC_m$  first, then  $\Delta_m = remain_m + \sum_{i \neq m} remain_i + deadline_i$ . If in another case, the scheduler chooses to schedule  $VC_n$  first, where  $n \neq m$ , then  $\Delta_n = remain_n + \sum_{i \neq n} remain_i + deadline_i$ . Thus  $\Delta_n = \Delta_m$  for all  $n \neq m$ . So it follows that the FCFS policy with the first incoming VC as the reference will produce the minimal total delay.  $\square$

We assume that all the frames have the same size in the above analysis. However, realistically different source traffic flows will most likely exhibit different bandwidth requirements. Thus in the next sub-section, we relax the constraint of having all frames from all VCs having the same size.

#### 3.2 With Different Frame Sizes

When the frame sizes are different, the selection of the contending frames from different VCs produces different values of total delay. Since the deadline is still common, the frame size is the only factor which is necessary to consider. Consider Figure 3, which is a variation of Case (c) of Figure 2 as shown in Section 1. This example demonstrates that there are two possible schedules after serving Source 3. One schedule is to use FCFS, which serves Source 1 first. The other schedule is to serve Source 2 first, which uses SJF (Shortest Job First) scheduling. As indicated by the total delay,  $\Delta$ , the FCFS scheduling results in a greater delay than the SJF scheduling.

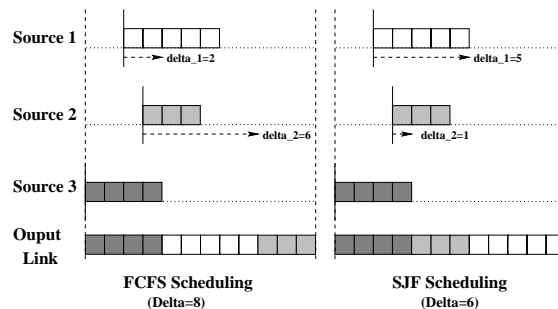


Figure 3: An example to demonstrate the failure of FCFS scheduling to guarantee the minimal total delay

**Lemma 2** Assuming all VCs have the same deadline,  $deadline_i$ , but not necessarily the same frame sizes, the SJF policy with the first incoming VC as the reference will produce the minimal total delay.

**Proof:** Without loss of generality, we assume that when the scheduler begins to choose a frame, all the  $N$  VCs have frames waiting for output. Thus, there are  $N!$  possible schedules to serve these  $N$  VCs. Assuming  $S = (s_1, s_2, s_3, \dots, s_N)$  is the schedule after sorting the frame sizes among these coming VCs, where  $s_i < s_j$  for all  $i < j$ , the total delay is the following.

$$\begin{aligned} \Delta_{SJF} &= 0 + s_1 + (s_1 + s_2) + \dots + (s_1 + s_2 + \dots + s_{N-1}) \\ &= (N-1) * s_1 + (N-2) * s_2 + \dots + (N-i) * s_i + \dots + \\ &\quad (N-j) * s_j + \dots + 2 * s_{N-2} + s_{N-1} \end{aligned}$$

Now assume there exists a scheduler where for some  $i$  and  $j$  such that  $s_i < s_j$  for  $i < j$ ,  $s_j$  is scheduled before

$s_i$ , resulting in a total delay less than  $\Delta_{SJF}$ . This is equivalent to interchanging the positions of  $s_i$  and  $s_j$  in the previous equation. Therefore, the new total delay is:

$$\Delta_{new} = (N-1) * s_1 + (N-2) * s_2 + \dots + (N-i) * s_j + \dots + (N-j) * s_i + \dots + 2 * s_{N-2} + s_{N-1}$$

It can be shown as the following that  $\Delta_{new} - \Delta_{SJF}$  is always greater than 0.

$$\begin{aligned} \Delta_{new} - \Delta_{SJF} &= (N-i) * s_j + (N-j) * s_i - \\ &\quad (N-i) * s_i - (N-j) * s_j \\ &= (i-j) * s_i + (j-i) * s_j \\ &= (j-i) * (s_j - s_i) \\ &> 0 \end{aligned}$$

because  $(j-i) > 0$  and  $(s_j - s_i) > 0$ .

Thus the new scheduler has a larger total delay than  $\Delta_{SJF}$  and a contradiction has been reached. Thus the SJF policy with the first incoming VC as the reference will produce the minimal total delay.  $\square$

We now propose a heuristic algorithm, which is a variation of SJF algorithm with the requirement that the first incoming VC is always served first. Lines 6-11 determine the frame with the smallest frame size and stores this VC in *saved\_VC*, if this data in any of the N sub-buffer-queues. Lines 13 and 14 compute the total delay and count the number of frames that have been served. Lines 17-21 detects the new incoming data and inserts them into their corresponding sub-buffer-queues.

**Algorithm for Different frame Sizes with a Common Deadline (DSCD):** Given  $N$  competing VCs with data pattern  $(s_i, deadline_i)$  where  $deadline_i = deadline_j$  for all  $i, j$ , find the frame with the shortest size among all the heads of the  $N$  sub-buffer-queues and compute the total delay,  $\Delta$ .

```

1 DSCD(N, VCi)
2 {
3   if (there is data in any of the N sub-buffer-queues)
4     /* MAX_SIZE is a constant larger than any frame sizes */
5     saved_size = MAX_SIZE;
6     for (i=1 to N)
7       if (si of the head of sub-queuei < saved_size)
8         saved_size = si;
9         saved_VC = i;
10      end if
11    end for
12    Move the head of sub-queuesaved_VC out of the queue;
13    Compute  $\Delta = \Delta + (curr\ ent\ time - start\ time)$ ;
14    served = served + 1;
15    Serve the frame on the output link;
16  end if
17  for (i=1 to N)
18    Scan for the new incoming data for VCi,
19    and record the start_time;
20    Insert the new incoming frame at the end of the sub-queuei;
21  end for
22 }
```

**Complexity Analysis:** With regard to complexity, the major difference in the DSCD algorithm is in Lines 5-

11 where we search for the frame with the smallest size. The time complexity of this algorithm is  $O(N)$ .

In this section all the VCs are assumed to have the same deadline requirement. However, in real-life applications it is most likely that different periodic sources will also have different deadlines<sup>2</sup>. In the next section, we assume different VCs can have different deadlines.

## 4 Periodic Sources with Different Deadlines

### 4.1 With the Same Frame Size and Different Deadlines

According to the results in Section 3.1, if the frames sizes of all the incoming VCs are the same, the total delay is fixed. However, besides the total delay, the number of frames which actually miss their deadline is also an important performance measure to applications with delay requirements. In order to minimize the number of frames that miss their deadlines, we need to consider how long a frame has been waiting inside the switch. Thus we extend the previously stated VC attributes,  $VC_i = (s_i, deadline_i)$ , to include another parameter,  $p_i$ , which denotes the number of cell slots that have elapsed since the arrival of the first cell in the current frame. Hence, each  $VC_i$  has attributes denoted by  $(s_i, deadline_i, p_i)$ . Note that:  $0 \leq p_i \leq deadline_i$ , if *frame<sub>i</sub>* has to meet its deadline. Also, the remaining number of cell slots left to transmit the frame is  $(deadline_i - p_i)$ .

Now, given a set of competing VCs, what is the criteria that should be used to schedule the VCs? Since the scheduling doesn't affect the total delay, our goal in this section has thus shifted to minimizing the number of frames which miss their deadlines. In Section 3.2 we showed that an earliest-deadline approach could be used to meet this goal. That is, at time  $t$ , choose the  $VC_i$  with the smallest value of  $(deadline_i - p_i)$ . In other words, pick the VC which is most likely to miss its deadline the soonest.

In the following algorithm, if there is data in any of the N sub-buffer-queues, lines 6-11 determine the frame with the minimal  $(deadline - p)$  and stores this VC in *saved\_VC*. Lines 13-16 count the total number of frames that miss their deadlines and the number of frames that have been served. Lines 19-22 detect the new incoming data and insert them into the corresponding sub-buffer-queues.

**Algorithm for Common frame Size with Different Deadlines (CSDD):** Given  $N$  competing VCs,

<sup>2</sup>For instance, a military officer sends out an urgent command and requires the message to be delivered to the destination as soon as possible.

each with an attribute  $(s_i, \text{deadline}_i, p_i)$  where  $s_i = s_j$  for  $i \neq j$ , find the frame  $i$  among all the heads of the  $N$  sub-buffer-queues, where frame  $i$  has the smallest  $(\text{deadline}_i - p_i)$ . Count the total number of frames that miss their deadlines,  $M$ .

```

1 CSDD( $N, VC_i$ )
2 {
3   if (there is data in any of the  $N$  sub-buffer-queues)
4     /* MAX_TIME has a value larger than any deadline */
5     saved_time = MAX_TIME;
6     for ( $i=1$  to  $N$ )
7       if  $((\text{deadline}_i - p_i)$  of the head of sub-queue $i < \text{saved\_time}$ )
8         saved_time =  $(\text{deadline}_i - p_i)$ ;
9         saved_VC =  $i$ ;
10      end if
11    end for
12    Move the head of subqueuesaved_VC out of the queue;
13    if  $((\text{deadline} - p) < 0)$ 
14       $M = M + 1$ ;
15    end if
16    served = served + 1;
17    Serve the frame on the output link;
18  end if
19  for ( $i=1$  to  $N$ )
20    Scan for new incoming data for  $VC_i$ ,
21    and record the start_time;
22    Insert the new incoming frame at the end of subqueue $i$ ;
23  end for
24 }
```

**Complexity Analysis:** The above algorithm is different from the DSCD algorithm in lines 5-11. The frame we want to find here is the one with the smallest  $(\text{deadline} - p)$  value. Since the time complexity for the loop from lines 6 to line 11 and the loop from line 19 to line 23 are both  $O(N)$ , the time complexity of this algorithm is thus also  $O(N)$ .

In Section 3, we considered the total delay of all the frames with the assumption that all the frames have the same deadline. In Section 4.1, we took into consideration another performance measure, the total number of frames that miss their deadlines. As mentioned earlier in Section 3, the most general case occurs when VCs have both different deadlines and different frame sizes. Given these conditions, both the total delay and the number of frames that miss their deadlines must be considered. The analysis for this general case is presented in the next subsection.

#### 4.2 With Different Deadlines and Different Frame Sizes

In order to achieve the minimum total delay, the algorithm proposed in Section 3.2 should be used. That is, the switch should always serve the frame with the smallest size so it will result in the minimum total delay. However, this may cause some large frames to wait too long and miss their deadlines. For an application, a frame missing its deadline will cause negative effects for the application, but a frame with a large delay, if it is still less than the deadline, will not necessarily adversely affect the application. So in this most general case, we propose to have our primary goal be to minimize the

number of frames which will miss their deadlines. Then, the second goal is to minimize the total delay.

To check if the selection of a frame,  $frame_i$ , will cause another frame  $frame_j$  to miss its deadline, we need to check the size,  $s_i$ , of  $frame_i$  with  $(\text{deadline}_j - p_j)$ , which is the maximum number of time slots  $frame_j$  can wait in the switch without missing its deadline. The service time of  $frame_i$  will increase  $p_j$  by  $s_i$ . So if  $s_i \leq (\text{deadline}_j - p_j)$ ,  $frame_j$  will not miss its deadline after  $frame_i$  is served first. Otherwise the deadline for  $frame_j$  will be missed. Now we provide a definition to be used to check this condition.

**Definition 2** Given two frames,  $frame_i$  and  $frame_j$ , if  $(\text{deadline}_i - p_i) \geq s_j$ ,  $frame_i$  is said to be **pairwise schedulable** to  $frame_j$ . This check is called the **pairwise schedulability check**.

In the following algorithm, lines 4-6 place all the heads of the  $N$  sub-buffer-queues into a set called *CURRENT* which contains all the frames that can be scheduled. Lines 11-17 use the *pairwise schedulability check* (defined above) to determine the set of frames, called *FIRST*, which are all the frames that are not *pairwise schedulable* to every frame. This set of VCs, *FIRST*, must be scheduled before the other VCs are served. Otherwise, they will cause some frames to miss their deadlines. Then, if there is more than one frame in *FIRST* and the size of *FIRST* has changed, the frames in the new set *FIRST* are checked using the *pairwise schedulability check* to determine, again, if there are frames within the set which must be scheduled before the others.

Lines 8-19 iteratively checks *FIRST*. Then *CURRENT* is used to store the *FIRST* set from the previous step. This process continues until (1) the *FIRST* set becomes empty, (2) there is only a single frame in the *FIRST* set, or (3) the size of the *FIRST* set doesn't change for 2 consecutive steps. Lines 20-21 handles the first case. This can occur in the following two sub-cases. First, the iteration to find *FIRST* has only been executed once, which means there is no *FIRST* set and no frame will cause other frames to miss their deadlines. Second, the iteration has been executed more than once, which means the *FIRST* set becomes an empty set at the last step. Thus a frame should be chosen from the previous *FIRST* set and it will cause the other frames in that set to miss their deadlines. For both of the sub-cases, we choose the frame with the minimal size in *CURRENT* to reduce the delay it will cause for other frames. Lines 22-23 handle the second case where the only VC in *FIRST* will be scheduled. The last case is handled in lines 24-25 and is the same as the second sub-case in case 2. We again choose the frame with minimal size. Then lines 27-30 compute the total delay and count the number of frames which miss their deadlines. Finally, lines 34-37 detect the new incoming

data and insert them into the corresponding sub-buffer-queues.

**Algorithm 1 for Different frame Size with Different Deadline (DSDD1):** Given  $N$  competing VCs, each with an attribute  $(s_i, \text{deadline}_i, p_i)$ , find the set *FIRST* using the pairwise schedulability check and choose the frame according to the size of *FIRST*. Compute the total delay,  $\Delta$ , and count the number of frames that miss their deadlines,  $M$ .

```

1 DSDD1( $N, VC_i$ )
2 {
3   if (there is data in any of the  $N$  sub-buffer-queues)
4     for ( $i=1$  to  $N$ )
5       Insert the head of sub-queue $_i$  to the set CURRENT;
6     end for
7     FIRST =  $\emptyset$ ;
8     repeat
9       oldsize = | FIRST |;
10      FIRST =  $\emptyset$ ;
11      for all  $VC_i$  such that ( $VC_i \in \text{CURRENT}$ )
12        for all  $VC_j$  such that
13          ( $VC_j \in \text{CURRENT}$ ) and ( $VC_i \neq VC_j$ )
14          if ( $(\text{deadline}_i - p_i) < s_j$ ) and ( $VC_i \notin \text{FIRST}$ )
15            put  $VC_i$  into the set FIRST;
16          end if
17        end for
18      end for
19      CURRENT = FIRST;
20    until (| FIRST | == oldsize) or (| FIRST | < 2);
21    if (| FIRST | == 0)
22      Move the frame with the smallest size
23      in CURRENT out of buffer queue;
24    else if (| FIRST | == 1)
25      Move the only frame in FIRST out of buffer-queue;
26    else /* (| FIRST | > 1) */
27      Move the frame with the smallest size
28      in FIRST out of buffer-queue;
29    end if
30    Compute  $\Delta = \Delta + (\text{current time} - \text{start\_time})$ ;
31    if ( $(\text{deadline} - p) < 0$ )
32       $M = M + 1$ ;
33    end if
34    served = served + 1;
35    Serve the frame on the output link;
36  end for
37  for ( $i=1$  to  $N$ )
38    Scan for new incoming data for  $VC_i$ ,
39    and record the start_time;
40    Insert the new incoming frame at the end of sub-queue $_i$ ;
41  end for
42 }
```

**Complexity Analysis:** The complexity of the DSDD1 algorithm is  $O(N^3)$  where  $N$  is the total number of VCs. This is because the **repeat ... until** loop (lines 9-20) will stop when the size of *FIRST* does not change or the size is less than 2. In the worst case, the size of the set *FIRST* goes to  $N$  and then decreases by one during each step of the **repeat ... until** loop. The loop stops when the size of *FIRST* is 1. Thus there are at most  $N$  steps for this loop. The creation of the *FIRST* set is performed by checking each pair of VCs for scheduling conflicts (lines 12-18). The complexity of this pairwise checking is  $O(N^2)$ , where  $N$  is the number of VCs being compared. Thus the complexity of the entire algorithm is  $O(N^3)$ .

### 4.3 Further Refinement

The algorithm to find the *FIRST* set can be improved. Instead of comparing each pair of frames in the set *CURRENT*, we can just compare the maximal size of the frames in *CURRENT* with all the other frames' ( $\text{deadline} - p$ ) values. When a frame's ( $\text{deadline} - p$ ) value is less than the maximal size in *CURRENT*, it will always be placed into *FIRST*, no matter what the results are of being compared with the other sizes. And finally we need to compare the ( $\text{deadline} - p$ ) value of the frame with the maximal size with the sizes of other frames to decide if this frame itself should be put to *FIRST*.

In the following algorithm, the only difference from the previous algorithm is lines 11-21, which determines the set *FIRST*. Line 11 determines the frame with the maximal size. Then lines 12-16 use this frame to check if all other frames are *pairwise schedulable* to this frame. Finally, the frame with the maximal size is checked to find out if it is *pairwise schedulable* to all the other frames from lines 17-21.

**Algorithm 2 for Different frame Sizes with Different Deadlines (DSDD2):** Given  $N$  competing VCs, each with an attribute  $(s_i, \text{deadline}_i, p_i)$ , find the set *FIRST* using the pairwise schedulability check and choose the frame according to the size of *FIRST*. Compute the total delay  $\Delta$  and count the number of frames that miss their deadlines,  $M$ .

```

1 DSDD2( $N, VC_i$ )
2 {
3   if (there is data in any of the  $N$  sub-buffer-queues)
4     for ( $i=1$  to  $N$ )
5       Insert the head of sub-queue $_i$  to the set CURRENT;
6     end for
7     FIRST =  $\emptyset$ ;
8     repeat
9       oldsize = | FIRST |;
10      FIRST =  $\emptyset$ ;
11      Find  $VC_i$  which has the maximal size in CURRENT;
12      for all  $VC_j$  such that ( $VC_j \in \text{CURRENT}$ )
13        ( $VC_j \neq VC_i$ )
14        if ( $\text{deadline}_j - p_j < s_i$ );
15          put  $VC_j$  into the set FIRST;
16        end if
17      end for
18      for all  $VC_j$  such that ( $VC_j \in \text{CURRENT}$ )
19        and ( $VC_j \neq VC_i$ )
20        if ( $\text{deadline}_i - p_i < s_j$ );
21          put  $VC_i$  into the set FIRST and break;
22        end if
23      end for
24      CURRENT = FIRST;
25    until (| FIRST | == oldsize) or (| FIRST | < 2);
26    if (| FIRST | == 0)
27      Move the frame with the smallest size
28      in CURRENT out of buffer queue;
29    else if (| FIRST | == 1)
30      Move the only frame in FIRST out of buffer-queue;
31    else /* (| FIRST | > 1) */
32      Move the frame with the smallest size
33      in FIRST out of buffer-queue;
34    end if
35    Compute  $\Delta = \Delta + (\text{current time} - \text{start\_time})$ ;
36    if ( $(\text{deadline} - p) < 0$ )
```

```

33     M = M + 1;
34   end if
35   served = served + 1;
36   Serve the frame on the output link;
37 end if
38 for (i=1 to N)
39   Scan for new incoming data for VCi, and
39   record the start_time;
40   Insert the new incoming frame to the end of subqueuei;
41 end for
42 }

```

**Complexity Analysis:** In this algorithm, lines 12-22 are used to find the *FIRST* set. The time complexity of this part has been improved to  $O(N)$ . Thus the time complexity of the entire algorithm is now  $O(N^2)$ .

## 5 Empirical Evaluation

### 5.1 Experiment Design

We created a scalable model to simulate an ATM switch with  $2^n$  input/output ports. The values of  $n$  ranged between 2 and 10 (i.e., up-to 1024 ports). As the results show later in this section, 32 ports are shown to be sufficient to demonstrate both the nature of the problem and the performance improvements provided by the proposed algorithms. The simulation is based on Ptolemy [18] Ver 0.7 for Linux operating system from University of California at Berkeley. Ptolemy is a flexible environment for a wide range of simulations. Various types of simulations can be built in different prototyping environments called *domains* in the system. The one used in our simulation is Discrete-Event (DE) domain, which is designed for simulating queueing systems, communication networks, and hardware systems.

For our experiments, we assumed the input sources were live sources. Hence we assumed a frame rate of 30 *fps*; the average period between 2 consecutive frames is 33 *msec*. The different data rates of each source then determine the frame sizes. In order to evaluate the results of the proposed algorithms we assume real-time periodic source traffic with bandwidths randomly chosen from the combinations of 1-*Mbps*, 2-*Mbps*, 4-*Mbps*, and 8-*Mbps*. In order to get a possible solution of this complex problem, worst-case constant bit rate is emulated in our experiments.

We also designed each experiment such that both the worst-case and the average-case performance results can be collected. We define the worst-case experiment scenario as the case where all the sources begin to transmit data at the same time and the time interval between two consecutive frames is the same for all sources. The average-case experiments are defined as when the sources randomly choose the transmission time of the frame within a period. Each experiment runs for 1 minute using actual traces from [7]. Due to the space

limitation, we only present the worst-case experiments in this paper.

### 5.2 Performance Improvements with DSCD Algorithm

Figure 4 illustrates the comparison of processing delay for the FCFS algorithm and the DSCD algorithm.

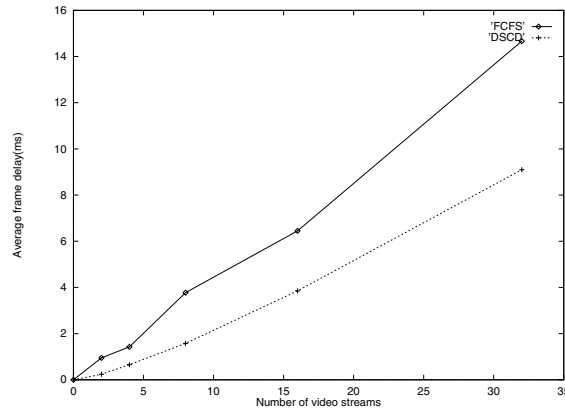


Figure 4: Performance Comparison of Processing Delays Using the Proposed DSCD Algorithm.

Figure 4 depicts the worst-case performance. Given the same number of periodic sources, our proposed DSCD algorithm has less processing delay than the conventional FCFS algorithm. For the case where the number of input ports is equal to 2, the performance of the FCFS algorithm has been improved from 0.951 *msec* to 0.238 *msec*. This result indicates approximately a 75.0% reduction in the processing delay from the FCFS algorithm. Similar trends in performance improvement also can be observed when the number of periodic sources is increased. For example, when the number of streams is increased from 2 to 4, a 54.2% reduction on the processing delay is achieved. 58.2%, 40.3%, and 38.0% reductions in processing delay are also achieved when the number of sources is doubled to 8, 16 and 32.

We have so far examined the performance improvements based on the assumptions that all source traffic is synchronized with the same deadline, which is implicitly applied with the same period. However, different deadlines across different sources can be quite common as previously discussed. We thus continue investigating the performance improvements in the next subsection with our CSDD algorithm, which is proposed mainly to address this issue.

### 5.3 Performance Improvements with CSDD Algorithm

In this subsection, we assume that the source frames can have different deadlines but the same frame size. When multiple frames are being scheduled, choosing any one of the frames will cause all the other frames to wait the same amount of time because they are all of the same size. In other words, the processing delay is not affected by the scheduling algorithms on this scenario. However, for an application, besides the processing delay, the number of source frames that miss their deadlines is also an important performance parameter.

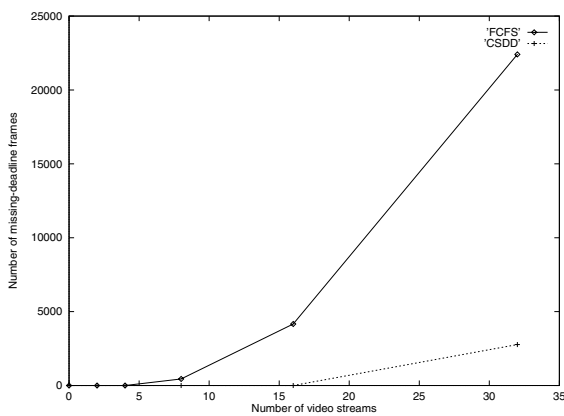


Figure 5 Performance Comparison of Number of Frames that Miss Deadline Using the Proposed CSDD Algorithm.

Figure 5 depicts the worst-case performance. We observe that when the number of streams is less than 8, neither of the two algorithms has frames that miss their deadlines. When there are 8 and 16 sources, the FCFS algorithm causes 445 frames and 4165 frames to miss their deadlines. In the mean time, the proposed CSDD algorithm has totally eliminated the number of frames that miss their deadlines. When the number of sources is increased to 32, the proposed CSDD algorithm generated an 88% reduction in the number of frames that miss their deadlines from the FCFS algorithm.

We assume that all the sources have the same frame size in this subsection. However, in real world applications usually have different frame sizes. A general scenario would be one in which sources can have both different frame sizes and different deadlines. The evaluation of this scenario is shown in the next subsection.

### 5.4 Performance Improvements with DSDD2 Algorithm

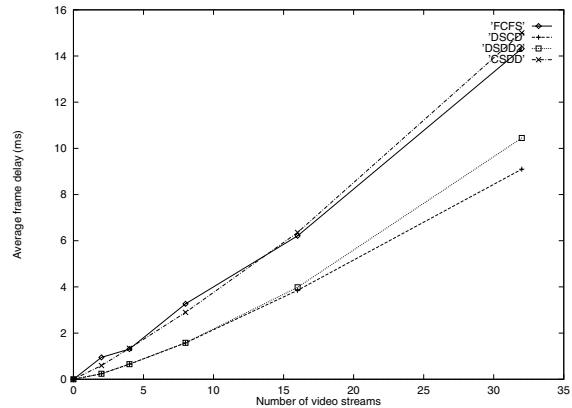


Figure 6 Performance Comparison of Proposed Algorithms on the Comparison of Processing Delays

Figure 6 depicts the worst-case performance comparison for processing delay. We observe that for all the four proposed algorithms, their delays increase when the number of periodic sources is increased. With 32 sources, the CSDD algorithm has the maximal delay among the four algorithms and the DSCD algorithm has the minimal delay. The delay of the FCFS algorithm is close to that of the CSDD algorithm; while the delay of the DSDD2 algorithm is close to that of the DSCD algorithm.

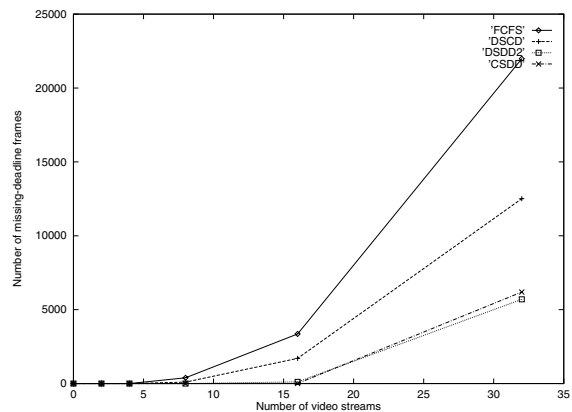


Figure 7 Performance Comparison of Proposed Algorithms on the Comparison of Number of Missed-Deadline Frames.

Figure 7 shows the worst-case performance comparison for number of missed-deadline frames. When the number of source streams is 2 or 4, no algorithm causes any frames to miss their deadline. With 8 source streams, both the FCFS algorithm and the DSCD algorithm begin to have miss-deadline frames. The CSDD

algorithm and DSDD2 algorithm begin to have missed-deadline frames when there are 16 source streams. With 32 source streams, the FCFS algorithm causes the maximal number of miss-deadline frames while the DSDD2 algorithm causes the minimal number. For the other two algorithms, the performance of the CSDD algorithm is close to that of the DSDD2 algorithm; the performance of the DSCD algorithm is worse than that of the CSDD algorithm, but better than that of the FCFS algorithm.

Considering both Figure 6 and Figure 7, it is obvious that the traditional FCFS algorithm is not suitable for mission-critical applications because it causes the maximal number of missed-deadline frames among the four algorithms and the delay of this algorithm is also large. The DSCD algorithm has the minimal delay but its performance for the number of missed-deadline frames is much worse than that of the CSDD and DSDD2 algorithm. The CSDD algorithm has a similar performance as the DSDD2 algorithm but it has the largest delay among the four algorithms.

Combining the total results from Figure 6 and Figure 7, we observe that the proposed DSDD2 algorithm outperforms the FCFS, DSCD and CSDD algorithms on balancing both the processing delay and the number of frames that miss deadlines.

## 6 Conclusion

In this paper, we have presented a systematic study of scheduling algorithms for a high-speed switch which must support mission-critical traffic flows. According to the nature of mission-critical applications, we focused our study on the frame-level performance. We defined two categories of traffic: (i) all frames have the same deadline and (ii) all frames have different deadlines. For the first category, when all the frame sizes are the same, we proved that the FCFS algorithm is optimal. An algorithm (i.e., DSCD) which can minimize the total delays with time complexity  $O(N)$  was given for the case of different frame sizes. For the second category, when all the frame sizes are the same, an algorithm (i.e., CSDD) with time complexity  $O(N)$  was proposed to minimize the number of frames that miss the deadlines. For the most general case of different deadlines with different frame sizes, we proposed an  $O(N^2)$  algorithm (i.e., DSDD2) that can minimize the number of missed-deadline frames with significant reduction in total processing delays. The experimental results of the proposed algorithms demonstrated that our proposed algorithms can achieve a significant improvement over the traditional FCFS algorithm. For the general case that frames can have both different sizes and different deadlines, the proposed DSDD2 algorithm thus provides the balanced performance improvement on both the number of frames that miss their deadlines and the total processing delays.

## References

- [1] H. Ahmadi and W. Denzel, "A Survey of Modern High-Performance Switching Techniques", *IEEE Journal on Selected Areas in Communications*, 7(7):1091-1103, Sep. 1989.
- [2] A. Dailianas and A. Bovopoulos, "Real-Time Admission Control Algorithms with Delay and Loss Guarantees in ATM Networks", *Proceedings of IEEE INFOCOM'94*, pp.1065-1072, 1994.
- [3] A. Dailianas and A. Bovopoulos, "Design of Real-Time Admission Control Algorithms with Priority Support", *Proceedings of IEEE INFOCOM'95*, pp.819-826, 1995.
- [4] N. Endo, T. Kozaki, T. Ohuchi, H. Kuwahara, and S. Gohara, "Shared Buffer Memory Switch for an ATM Exchange", *IEEE Transactions on Communications*, 41(1):237-245, Jan. 1993.
- [5] N. Figueira and J. Pasquale, "An Upper Bound on Delay for the VirtualClock Service Discipline", *IEEE/ACM Transactions on Networking*, 3(4):399-408, Aug. 1995.
- [6] D. Ferrari and D. Verma, "A Scheme for Real-Time Channel Establishment in Wide-Area Networks", *IEEE Journal on Selected Areas in Communications*, 8(3):368-379, Apr. 1990.
- [7] A. Guha, A. Pavan, J. Liu, A. Rastogi, and T. Steeves, "Supporting Real-Time and Multimedia Applications on the Mercuri Testbed", *IEEE Journal on Selected Areas in Communications*, 13(4):749-763, May 1995.
- [8] M. Hluchyj and M. Karol, "Queueing in High-Performance Packet Switching", *IEEE Journal on Selected Areas in Communications*, 6(9):1587-1597, Dec. 1988.
- [9] M. Hashemi and A. Leon-Garcia, "A General Purpose Cell Sequencer/Scheduler for ATM Switches", *Proceedings of IEEE INFOCOM'97*, 1997.
- [10] D. Heyman, A. Tabatabai, and T. Lakeshman, "Statistical Analysis and Simulation Study of Video Conference Traffic in ATM Networks", *IEEE Transactions on Circuits and Systems for Video Technology*, 2(1):49-59, Mar. 1992.
- [11] M. Karol, M. Hluchyj, and S. Morgan, "Input Versus Output Queueing on a Space-Division Packet Switch", *IEEE Transactions on Communications*, 35(12):1347-1356, Dec. 1987.
- [12] C. Li, Y. Ofek, A. Segall, and K. Sohraby, "Pseudo-Isochronous Cell Switching in ATM Networks", *Proceedings of IEEE INFOCOM'94*, pp.428-437, 1994.
- [13] J. Liu, D. Du, and J. Schnepf, "Supporting random access on real-time retrieval of digital continuous media", *Computer Communications*, 18(3):145-159, Mar. 1995.
- [14] D. Stiliadis and A. Varma, "Design and Analysis of Frame-based Fair Queueing: A New Traffic Scheduling Algorithm for Packet-Switched Networks", *Proceedings of SIGMETRICS'96*, pp.104-115, 1996.
- [15] J. Stankovic, M. Spuri, M. DiNatale and G. Buttazzo, "Implications of Classical Scheduling Results for Real-Time Systems", *IEEE Computer*, 28(6), 16-25, Jun. 1995.
- [16] D. Stiliadis and A. Varma, "Efficient Fair Queueing Algorithms for Packet-Switched Networks", *IEEE/ACM Transactions on Networking*, 6(2):175-185, 1998.
- [17] L. Zhang, "VirtualClock: A New Traffic Control Algorithm for Packet-Switched Networks", *ACM Transactions on Computer Systems*, 9(2):101-124, May 1991.
- [18] <http://ptolemy.eecs.berkeley.edu/>