

# QoS Provisioning and Tracking Fluid Policies in Input Queueing Switches

Vahid Tabatabaee<sup>1</sup>, Leonidas Georgiadis<sup>2</sup>, Leandros Tassioulas<sup>1</sup>

<sup>1</sup>Department of Electrical and Computer Engineering and Institute for Systems Research  
University of Maryland at College Park

<sup>2</sup>Electrical and Computer Engineering Department  
Aristotle University, Thessaloniki, Greece

**Abstract**—The concept of tracking policies for fluid policies is extended to the input queueing switches. It is considered that the speed up of the switch is one. For the special case of  $2 \times 2$  switches it is shown that tracking policy always exists. One of the interesting applications of the tracking policy in TDMA Satellite Switches is elaborated. For the general case of  $N \times N$  switches a heuristic tracking policy is provided. The heuristic algorithm is based on two notions of port tracking and critical links. These notions can be employed in derivation of other heuristic tracking policies as well. Simulation results present the usefulness of the heuristic algorithm and the two basic concepts it relies on.

## I. INTRODUCTION

One of the main issues in the design of integrated service networks is to provide the service performance requirements for a broad range of applications. Applications requirements are translated into network quantitative parameters. The most common performance measures are packet loss probability, delay, and jitter. The delay and jitter characteristics at each switch of the network is determined by the scheduling algorithm used in the switch and the incoming traffic pattern. On the other hand, the network should also be capable to analyze the amount of resources that each particular application requires. Based on this analysis a connection request is admitted or rejected. It is therefore, very important for the network designer to understand the effect a scheduling policy has on the connection performance and on the usage of network resources.

In many cases, it is easier to perform the analysis and design of scheduling policies under the modeling assumption that the traffic arrives and is treated as a fluid, i.e., the realistic case where information is organized into packets is not taken into account, [3],[4],[11],[12],[6]. Under the fluid policy, we assume that at every time instant arbitrary fractions of the link capacity can be shared among different applications. Although in most of the practical situations this is an idealistic assumption, it enables us to analyze the effect of scheduling policy on the network resources as well as the major performance parameters, and therefore to design the scheduling policies more conveniently. One approach to the design of packetized policies is to first find an appropriate fluid policy, and then to derive a packetized policy that resembles or tracks the fluid policy in a certain sense.

Existence of packetized tracking policies is a well established fact in the single link case. In fact, several tracking policies are suggested and their performance and efficiency are analyzed [11], [12], [6], [5], [9]. However, the existence of such

policies in input queueing switches is still an open problem. This is the main subject of this paper.

The research on scheduling  $N \times N$  switches is mainly concentrated on output queueing switches. In an  $N \times N$  switch, it is possible that all  $N$  inputs have packets for the same output at the same time. In order to accommodate such a scenario in an output queueing switch, the switch fabric should work  $N$  times faster than the line rates. This might be acceptable for moderate size switches working on moderate line rates, but as the capacity of the lines as well as the switch sizes increase, memories with sufficient bandwidth are not available and input queueing is becoming a more attractive alternative.

One way to circumvent this problem is to have Combined Input Output Queueing (CIOQ) switches with limited speed up that matches the output sequence of a purely output queueing switch. In fact, it is shown in [2] that speed up of 2 is sufficient to resemble the output pattern of any output queueing switch. However, the scheduling algorithm proposed to do that is fairly complicated, and the arbiter still requires to receive information from the input ports of the switch with speed up of  $N$ .

In this paper we consider an input queueing switch, where every input and output port can service 1 packet per time unit (All packets are considered to have equal size). In a fluid policy model, at every time slot every input (output) port can be connected to several output (input) ports, however the total service rate of any port should not exceed its capacity. Under a packetized policy, every input (output) port can at most be connected to one output (input) port at every time slot, i.e., there is no speed up in the switch fabric. Under these circumstances, our objective is to find a packetized policy that tracks a given fluid policy in an appropriate manner. For the special case of  $2 \times 2$  switches the existence of tracking policies is proved and a non-anticipative tracking policy is provided. For the general case a heuristic algorithm with good, but not perfect tracking properties is proposed. In fact, in the simulations done, less than 1 percent of the packets lost track of the fluid policy, when the utility of the switch is around 92%.

Another interesting application of the tracking policies is in the scheduling of TDMA Satellite Switches (TDMA-SS) with multi-periodic messages. In this problem the objective is to schedule a packet during every period of a connection stream, and before arrival of the next packet. Since it is not usually possible to queue the packets in the satellite switches, an input

queueing model is more appropriate in this case. The fluid policy that accomplishes the specified TDMA objective is trivial. The original problem is then solved by specifying a packetized policy that tracts that fluid policy.

The organization of paper is as follows. In the next section, we will review the concepts of fluid and tracking policies, and provide the feasibility condition for both cases. The problem of scheduling multi-periodic messages in TDMA-SS is explained and elaborated in section III. We will show that this problem is essentially a special case of the posed input queueing scheduling problem. In section IV, we show that for the  $2 \times 2$  switches a tracking policy always exist, and we provide a non-anticipative algorithm to find the tracking policy. In this section, we also address the problem of providing a packetized policy that satisfies pre-specified packet deadlines. In section V, some useful ideas regarding the design of heuristic tracking policies are given. Based on these concepts a heuristic simulation algorithm is proposed. The heuristic algorithm is applied to the scheduling of a multi-periodic TDMA-SS and the simulation results are given.

## II. FLUID AND PACKETIZED TRACKING POLICIES

We consider input queueing switches that serve fixed size packets. Each input and output port has the capacity of serving 1 packet per time unit. Since queues exist only at the input ports, the latter assumption implies that traffic of at most 1 packet per unit of time can be transferred from the input ports to a given output port.

We assume that the time is slotted and the length of each slot is equal to the length of a packet. Slots are numbered starting from 1, 2, ... . Slot  $k$  is taking the time interval  $(k - 1, k]$ . Time  $k - 1$  ( $k$ ) is the beginning (end) of time slot  $k$ . Packets arrive at the beginning of each time slot. Packets with origin input port  $i$  and destination output port  $j$  are served FCFS.

Two broad classes of policies are considered, the fluid and the packetized policies. During time slot  $k$  a fluid policy transmits,  $w_{ij}(k) \geq 0$ , units of information from input port  $i$  to output port  $j$ .  $w_{ij}(k)$  is a nonnegative real number and is measured in units of packets. Since at most one unit of work can be transferred from a given input port to the output ports, and since no queueing is permitted at the output ports, the  $w_{ij}(k)$ 's must satisfy the following inequalities.

$$\begin{aligned} w_{ij}(k) &\geq 0 \\ \sum_j w_{ij}(k) &\leq 1, \quad \forall i \in \{1, \dots, N\} \\ \sum_i w_{ij}(k) &\leq 1, \quad \forall j \in \{1, \dots, N\} \end{aligned} \quad (1)$$

A packetized policy is based on the assumption that during a time slot an input port can transmit a single packet to any one of the output ports. Therefore, for a packetized policy we have that  $J_{ij}(k)$ , the number of packets transmitted from port  $i$  to port  $j$  during slot  $k$ , is either 0 (no packet transmission during slot  $k$ ) or 1 (a single packet transmission during slot  $k$ ). A packetized policy is feasible if at every time slot  $k$  we have,

$$\begin{aligned} \sum_j J_{ij}(k) &\leq 1, \quad \forall i \in \{1, \dots, N\} \\ \sum_i J_{ij}(k) &\leq 1, \quad \forall j \in \{1, \dots, N\} \\ J_{ij}(k) &\in \{0, 1\}. \end{aligned} \quad (2)$$

Note that the conditions in (2) imply that for any  $k$ , there can be at most a single 1 in each column or row of the matrix  $[J_{ij}(k)]$ . That is, the matrix  $J_{ij}(k)$  is a sub-permutation matrix.

Usually fluid policies cannot be applied directly in a network since mixing of traffic belonging to different packets is not allowed. However, they are considered in this paper, because the performance analysis and the scheduling policy design is often more convenient for fluid policies. An approach to the design of packetized policies is to first design and analyze a fluid policy, and then implement a packetized policy that resembles in a certain sense the departure process of the fluid policy. Such a packetized policy is called a tracking policy. More precisely, for our purposes, we use the following definition:

**Definition I:** Given a fluid policy  $\pi$ , we say that a packetized policy is tracking  $\pi$  if every packet departs under the packetized policy at the latest by the end of the time slot at which the same packet departs under the fluid policy.

A basic question is if tracking policies exist for a given fluid policy. This question is answered positively for the single link case, where different sessions share a single link [11], [6]. In that case, perhaps the most well known fluid policies are GPS and rate controlled service disciplines. Several tracking policies are suggested for the single link case [11], [6], [9], [1]. The same concepts of GPS and rate controlled schedulers can be extended to the multi-input, multi-output input queueing switches. However, the existence of tracking policies for these switches is still an open question.

Searching for a tracking policy can be converted to another scheduling problem, scheduling of packets with deadlines. Suppose that a set of packets are given, every packet has two associated time stamps. The first time stamp is the eligible time, which is the time after which we can schedule the packet. For instance, this can be the arrival time of the packet to the switch. The second time stamp is the deadline of the packet. The objective is to schedule a packet inside the time frame of eligibility time and deadline time. Obviously, if a packetized scheduling policy satisfies all the deadlines induced by a fluid policy, it is a tracking packetized policy by our definition.

In section IV, we study the special case of  $2 \times 2$  switches. We will prove that for the special case of  $2 \times 2$  switches, for every feasible fluid policy there exists a feasible packetized policy. In fact, our proof is constructive and will provide an algorithm to derive a tracking policy. We will also show that a natural extension of the earliest deadline first scheduling can be used to solve the problem of scheduling packets with deadline. The general case of an  $N \times N$  input queueing switch is currently under investigation. For the latter case, we provide in this paper a heuristic algorithm which shows very good performance under a number of simulation studies.

### III. MULTI-PERIODIC TDMA SATELLITE SWITCHES

One of the potential applications of tracking policies is in the scheduling of TDMA Satellite Switches (TDMA-SS). The conventional method to do the scheduling is based on the Inukai method [10]. This method is based on the assumption that all messages have the same period. The scheduling is done for a frame length equal to the period of the messages and it is repeated periodically thereafter. Let  $L$  be equal to the maximum number of packets that can be serviced by an input/output port during one period. A set of messages is schedulable if for every port the total number of packets that should be serviced is not more than  $L$ . Inukai has provided a scheduling algorithm for any set of schedulable messages.

The Inukai algorithm does not work appropriately when messages have different periods. Let message  $m$  from input port  $s_m$  to output port  $d_m$  has period  $p_m$ . To apply the Inukai method the frame length should be set to the LCM of all message periods, say  $L$ . For each message  $m$ ,  $L/p_m$  unit length packets are scheduled in the frame. Each of these packets is associated to one period of the original message. Then, we can use the Inukai method to allocate these packets inside the frame length  $L$ . The problem is that there is no control over the place of packets inside the frame in the Inukai method. Thus, it is possible that all packets attributed to a single periodic message are placed next to each other. Such an assignment suffers from high jitter. Moreover, the delay of a packet can be equal to  $L$ , which can be very large.

Suppose that the objective is to schedule every packet in the time frame of its period. Thus, every packet can tolerate a delay up to its period. The question then arises whether it is possible to provide a schedule under these constraints. A necessary condition for schedulability, is that the utilization of every input port  $i$  and output port  $j$  should not be greater than unity, i.e.,

$$\begin{aligned} u_i &= \sum_{m:s_m=i} \frac{1}{p_m} \leq 1, \\ u_j &= \sum_{m:d_m=j} \frac{1}{p_m} \leq 1, \end{aligned} \quad (3)$$

If one considers fluid policies, then it is easy to provide a schedule provided that (3), is satisfied. Specifically, consider the fluid policy that assigns the exact service rate of  $1/p_m$  to every message  $m$ . Under this policy the switch starts servicing every packet immediately after its arrival, and it takes  $p_m$  time units to complete its service. This means that the target deadlines are all accomplished. Therefore, if we can provide a packetized policy that tracks the fluid policy, then this packetized policy will satisfy the delay constraints as well, and (3) becomes the necessary and sufficient condition for the schedulability of packetized policies under the specified constraints.

In [13] Philp and Liu conjectured that (3) is the necessary and sufficient condition for schedulability under the specified

delay constraints. Giles and Hajek [8] have proved this conjecture for a special case. In their model the messages are sorted based on their period, such that,

$$p_1 \geq p_2 \geq \dots \geq p_M.$$

Moreover, for every two subsequent messages we have,

$$p_m = kp_{m+1},$$

where  $k$  is an integer. Unfortunately, their algorithm does not work well in the general case.

As we saw, the correctness of the conjecture is trivial for the fluid policies, but it has not been proved for packetized policies. In the next section, we show the existence of tracking policies for the special case of  $2 \times 2$  switches. Thus, the conjecture is proved for the special case of  $2 \times 2$  switches.

### IV. THE $2 \times 2$ SWITCH

In this section, we consider a  $2 \times 2$  input queueing switch and provide an algorithm for designing a packetized policy  $\pi_i$  that tracks a given fluid policy  $\pi$ .

Let  $W_{ij}(k)$ ,  $k = 0, \dots, L$  be the total amount of traffic transferred under the fluid policy, from input port  $i$  to output port  $j$  by the end of slot  $k$ . Thus, we have,

$$W_{ij}(k) = \sum_{l=1}^k w_{ij}(l), \quad (4)$$

where  $w_{ij}(k)$  satisfy conditions (1).

In the following, we address the following problem. Problem I: Find a sequence of sub-permutation matrices  $J(1), J(2), \dots$ , such that for all  $k$  we have

$$\lfloor W_{ij}(k) \rfloor \leq I_{ij}(k) < W_{ij}(k) + 1, \quad i = 1, 2, \quad j = 1, 2, \quad (5)$$

where

$$I_{ij}(k) = \sum_{l=1}^k J_{ij}(l).$$

If a solution to Problem I can be found, then the packetized policy that uses the sub-permutation matrix  $J(k)$  to schedule packets during slot  $k$  is tracking the given fluid policy. To see this, notice first that according to the leftmost inequality in (5), by the end of slot  $k$ , if a (integer) number of packets completes transmission between input port  $i$  and output port  $j$  under the fluid policy, then at least the same number of packets completes transmission between the same input and output port under the packetized policy. This and the fact that packets between ports  $i$  and  $j$  are served FCFS under both policies, imply that: If a packet completes transmission in slot  $l$  under the fluid policy, it will also complete transmission at the latest by the end of slot  $l$  under the packetized policy.

We note next that a realizable packetized policy should serve packets after their arrival to the switch. This is ensured by the

rightmost inequality in (5). To see this, notice that according to this inequality, the number of packets that complete transmission between input  $i$  and output port  $j$  under the packetized policy cannot exceed the number of packets that complete transmission between  $i$  and  $j$  under the fluid policy by more than 1. Moreover, if  $W_{ij}(k)$  is integer then necessarily,  $W_{ij}(k) = I_{ij}(k)$ . This and the fact that packets are served FCFS under both policies, imply that:

A packet cannot complete transmission in slot  $k$  under the packetized policy unless part of this packet has already been transmitted up to the end of slot  $k$  by the fluid policy.

Since the fluid policy is feasible and never begins transmission of a packet before its arrival, the same holds for the packetized policy.

Note that if we solve Problem I, we in fact have a packetized policy that not only tracks the finishing times of the packets under the fluid policy, but also the times when the fluid policy starts transmission of these packets.

Before we proceed we need another definition.

**Definition II:** An integer valued matrix  $I$  is called a u-neighbor of a matrix  $W$  if

$$\lfloor W_{ij} \rfloor \leq I_{ij} < W_{ij} + 1 \text{ for all } i \text{ and } j, \quad (6)$$

$$\left\lfloor \sum_{i=1}^2 W_{ij} \right\rfloor \leq \sum_{i=1}^2 I_{ij} \text{ for all } j, \quad (7)$$

and

$$\left\lfloor \sum_{j=1}^2 W_{ij} \right\rfloor \leq \sum_{j=1}^2 I_{ij} \text{ for all } i. \quad (8)$$

We now address a stricter version of Problem I.

**Problem II:** Find a sequence of sub-permutation matrices  $J(1), J(2), \dots$ , such that for all  $k$ ,  $I(k)$  is a u-neighbor of  $W(k)$ , where

$$I_{ij}(k) = \sum_{i=1}^k J_{ij}(k).$$

We now proceed to provide a solution for Problem II. The proof is by induction. At the beginning of slot 0 no traffic has been processed under either of the policies (fluid and packetized), and we set

$$W(0) = I(0) = [0].$$

Assume now that we have found appropriate sub-permutation matrices until slot  $k$ . We show how to construct the sub-permutation matrix for slot  $k+1$ ,  $J(k+1)$ , based on  $I(k)$  and  $W(k+1)$ , so that the matrix  $I(k) + J(k+1)$  is a u-neighbor of  $W(k+1)$ .

Set

$$J_{ij}(k+1) = \max \{ \lfloor W_{ij}(k+1) \rfloor - I_{ij}(k), 0 \}. \quad (9)$$

$J(k+1)$  is a sub-permutation matrix. To see this note first that

$$\begin{aligned} \lfloor W_{ij}(k+1) \rfloor - I_{ij}(k) &= \\ \lfloor W_{ij}(k) + w_{ij}(k+1) \rfloor - I_{ij}(k) &\leq \\ \lfloor W_{ij}(k) \rfloor - I_{ij}(k) + 1 &\leq 1 \end{aligned} \quad \text{by (6).}$$

The second inequality holds because  $w_{ij}(k+1) \leq 1$ . Therefore,  $J_{ij}(k+1) = 0$  or 1. Next we have to show that there can be no more than a single 1 in each column or row of  $J(k+1)$ . To see this, note that if  $J_{ij}(k+1) = 1$ , then by (9)  $I_{ij}(k) + 1 = \lfloor W_{ij}(k+1) \rfloor$ , and therefore,

$$\begin{aligned} W_{ij}(k) + w_{ij}(k+1) - I_{ij}(k) &= \\ W_{ij}(k+1) - I_{ij}(k) &\geq \\ \lfloor W_{ij}(k+1) \rfloor - I_{ij}(k) &= 1. \end{aligned}$$

Hence, if there is more than a single 1 in, say, column 1 of  $J(k)$ , then

$$\begin{aligned} \sum_{i=1}^2 W_{i1}(k) - \sum_{i=1}^2 I_{i1}(k) &\geq 2 - \sum_{i=1}^2 w_{i1}(k+1) \geq 1 \\ &\text{since } \sum_{i=1}^2 w_{i1}(k+1) \leq 1. \end{aligned}$$

But the last inequality contradicts the fact that

$$\begin{aligned} \sum_{i=1}^2 W_{i1}(k) &< \left\lfloor \sum_{i=1}^2 W_{i1}(k) \right\rfloor + 1 \\ &\leq \sum_{i=1}^2 I_{i1}(k) + 1 \end{aligned} \quad \text{by (7).}$$

We now show that  $I(k+1)$  satisfies (6). From (9) we have that

$$I_{ij}(k+1) = I_{ij}(k) + J_{ij}(k+1) \geq \lfloor W_{ij}(k+1) \rfloor$$

Also, if  $J_{ij}(k+1) = 0$

$$\begin{aligned} I_{ij}(k+1) &= I_{ij}(k) \\ &< W_{ij}(k) + 1 \quad \text{by (6)} \\ &\leq W_{ij}(k+1) + 1 \quad \text{since } w_{ij}(k+1) \geq 0. \end{aligned}$$

while if  $J_{ij}(k+1) = 1$ ,

$$\begin{aligned} I_{ij}(k+1) &= I_{ij}(k) + J_{ij}(k+1) \\ &= \lfloor W_{ij}(k+1) \rfloor \quad \text{by (9)} \\ &< W_{ij}(k+1) + 1. \end{aligned}$$

It remains to prove (7) and (8) for the matrix  $I(k+1)$ . Consider, say, column 1. We distinguish the following cases.

Case 1.  $J_{i1}(k+1) = 1$  for some  $i$ . Then  $\sum_{i=1}^2 J_{i1}(k+1) \geq 1$  and hence

$$\begin{aligned} \sum_{i=1}^2 I_{i1}(k+1) &= \\ \sum_{i=1}^2 (I_{i1}(k) + J_{i1}(k+1)) &\geq \\ \left\lfloor \sum_{i=1}^2 W_{i1}(k) \right\rfloor + 1 &= \\ \left\lfloor \left( \sum_{i=1}^2 W_{i1}(k) \right) + 1 \right\rfloor &\geq \\ \left\lfloor \sum_{i=1}^2 (W_{i1}(k) + w_{i1}(k+1)) \right\rfloor &= \\ \left\lfloor \sum_{i=1}^2 W_{i1}(k+1) \right\rfloor. & \end{aligned}$$

The third relation above holds because of (7). The fourth relation is correct since  $\lfloor \alpha \rfloor + m = \lfloor \alpha + m \rfloor$ ,  $m$  integer, and the fifth relation follows from  $\sum_{i=1}^2 w_{i1}(k+1) \leq 1$ .

Case 2.  $J_{11}(k+1) = J_{21}(k+1) = 0$ . Then by (9)  $I_{i1}(k) \geq \lfloor W_{i1}(k+1) \rfloor$ ,  $i = 1, 2$ . Since we also have  $I_{i1}(k) < W_{i1}(k) + 1 \leq W_{i1}(k+1) + 1$ , we conclude that the difference  $I_{i1}(k) - \lfloor W_{i1}(k+1) \rfloor$  can take only the values 0 and 1. We now need to distinguish the following sub-cases.

Sub-case 2.1  $I_{i1}(k) = \lfloor W_{i1}(k+1) \rfloor + 1$  for some  $i$ . Then

$$\begin{aligned} \sum_{i=1}^2 I_{i1}(k+1) &\geq \sum_{i=1}^2 I_{i1}(k) \\ &\geq \sum_{i=1}^2 \lfloor W_{i1}(k+1) \rfloor + 1 \\ &\geq \left\lfloor \sum_{i=1}^2 W_{i1}(k+1) \right\rfloor \end{aligned}$$

The last relation is correct since  $1 + \lfloor \alpha \rfloor + \lfloor \beta \rfloor \geq \lfloor \alpha + \beta \rfloor$ .

1. Sub-case 2.2  $I_{i1}(k) = \lfloor W_{i1}(k+1) \rfloor$  for all  $i$ .

If  $W_{i1}(k+1)$  is integer for some  $i$ , then

$$\begin{aligned} \left\lfloor \sum_{i=1}^2 W_{i1}(k+1) \right\rfloor &= \sum_{i=1}^2 \lfloor W_{i1}(k+1) \rfloor \\ &= \sum_{i=1}^2 I_{i1}(k) \\ &= \sum_{i=1}^2 I_{i1}(k+1) \end{aligned}$$

The last relation follows since  $J_{i1}(k+1) = 0$ ,  $i = 1, 2$ . It remains to consider the case where  $W_{i1}(k+1)$  is non-integer

for all  $i$ . We may then redefine  $J_{i1}(k+1) = 1$  for some  $i$  and still have a sub-permutation matrix. To see this, consider the following two cases

(a) In column 2 there is already  $J_{i2}(k+1) = 1$  for some  $i$ , say  $i = 1$ . Then, since  $J(k+1)$  is a sub-permutation matrix, we know that  $J_{22}(k+1) = 0$ . Therefore, setting  $J_{21}(k+1) = 1$  we still have a sub-permutation matrix  $J(k+1)$ .

(b)  $J_{i2}^{k+1} = 0$  for all  $i$ . Then we can set  $J_{i1}^{k+1}$  for one of the  $i$ 's and still have a sub-permutation matrix  $J(k+1)$ .

In order to show that (6) still holds with the modified  $J(k+1)$ , note that since  $I_{i1}(k) = \lfloor W_{i1}(k+1) \rfloor$  and  $W_{i1}(k+1)$  is non-integer,

$$I_{i1}(k+1) = \lfloor W_{i1}(k+1) \rfloor + J_{i1}(k+1) < W_{i1}(k+1) + 1$$

and clearly,

$$\lfloor W_{i1}(k+1) \rfloor \leq I_{i1}(k) + J_{i1}(k+1) = I_{i1}(k+1).$$

To show that (8) holds, note that since for the modified matrix we now have  $J_{i1}(k+1) = 1$ , we can apply the same argument as in case 1 above.

We may continue in this fashion and examine the rest of the columns and rows, and update, if necessary, the matrix  $J^{k+1}$ . We eventually will have that the matrix

$$I^{k+1} = I_{i1}^k + J_{i1}^{k+1}$$

is a  $u$ -neighbor of  $a^{k+1}$ .

According to the procedure above, we have the following algorithm for creating the tracking policy, i.e., for generating the sub-permutation matrix  $J(k+1)$ .

ALGORITHM

1. At the beginning of slot  $k+1$  create the matrix elements  $J_{ij}(k+1)$  for  $i, j = 1, 2$  as follow,

$$J_{ij}(k+1) = \max \{ \lfloor W_{ij}(k+1) \rfloor - I_{ij}(k), 0 \}$$

2. If for some column or row, say column 1, it holds

$$\begin{aligned} I_{i1}(k) &= \lfloor W_{i1}(k+1) \rfloor \text{ for all } i \text{ and} \\ W_{i1}(k+1) &\text{ is non-integer for all } i, \end{aligned}$$

then redefine  $J_{i1}(k+1) = 1$  so that the modified matrix  $J(k+1)$  is still a sub-permutation matrix.

Note that the policy obtained in this way is non-anticipative, i.e., it does not depend on future arrivals and therefore, can be implemented on-line.

So far, we have shown that the tracking policy exists, and a procedure to convert a fluid policy to a packetized tracking policy is given.

In the previous section, it was mentioned that to obtain a tracking policy, we can convert the problem to a deadline scheduling problem and solve that problem. In the following a simple extension of the EDF algorithm for a  $2 \times 2$  switch is given. This policy, which we call it the EDF2 always come up with an admissible schedule if such a schedule exists. A schedule is admissible if it satisfies all the deadlines.

### A. Per-flow Tracking

So far, we have considered the per-link tracking policies, and proved their existence for  $2 \times 2$  switches. Basically, we showed that for every fluid policy there exists a packetized policy that tracks the aggregate traffic going from every input port  $i$  to every output port  $j$ . In many circumstances this is not quite enough.

The aggregate traffic between any pair of input/output nodes consists of several distinct flows or sessions. Generally, to provide per-flow QoS, the fluid policy should work on the granularity of flows and guarantees the service rate given to every flow individually. This should be also reflected in the packetized tracking policy. More specifically, the corresponding packetized policy should track the service given to every flow under the fluid policy.

Let  $W_{ij}^l(k)$  and  $I_{ij}^l(k)$  be the total amount of flow  $l$  traffic transferred from port  $i$  to  $j$  up to time  $k$  under the fluid policy,  $\pi_f$ , and the packetized policy,  $\pi_p$ , respectively. Then, the packetized policy  $\pi_p$  is a per-flow tracking policy of  $\pi_f$  if and only if,

$$[W_{ij}^l(k)] \leq I_{ij}^l(k) \leq [W_{ij}^l(k)] \quad \forall i, j, k, l.$$

Obviously, this implies a stricter definition for tracking policies. Recently, we were able to prove this notion of tracking policy for  $2 \times 2$  switches as well. The result is given without proof in the following theorem.

**Theorem 1:** Consider any arbitrary fluid policy  $\pi_f$  for input queueing  $2 \times 2$  switches. There exists a non-anticipative packetized policy  $\pi_p$  that tracks the individual services given to every flow under the fluid policy  $\pi_f$ .

### B. QoS provisioning in $2 \times 2$ switches

An alternative approach to the QoS provisioning problem is to view directly the requirements of the real-time traffic and to attempt to satisfy them. A natural framework for this approach is deadline satisfaction. Every packet presents upon arrival the maximum waiting time that may tolerate and a deadline is determined by which it should depart. The scheduler attempts to satisfy as many deadlines as possible. For a single link it is known that the Earliest Deadline First policy satisfies the packet deadlines if those are satisfiable. We show here that the same effect is achievable in  $2 \times 2$  switches if the same notion is generalized.

For each link  $(i, j)$  at every slot  $t$  let  $D_{ij}(t)$  be the earliest deadline among the backlogged packets. There are two service configurations  $o = \{(1, 1), (2, 2)\}$ ,  $x = \{(1, 2), (2, 1)\}$ . Let  $D^o(t)$  and  $D^x(t)$  be the sorted deadline vectors associated to the two configurations, such that the first component always be the minimum deadline, i.e.,

$$D^o(t) = (\min \{D_{11}(t), D_{22}(t)\}, \max \{D_{11}(t), D_{22}(t)\}),$$

$$D^x(t) = (\min \{D_{12}(t), D_{21}(t)\}, \max \{D_{12}(t), D_{21}(t)\}).$$

We say that the deadline vector  $D^i$  is lexicographically smaller than  $D^j$  if and only if,

$$(D_1^i < D_1^j) \text{ or}$$

$$(D_1^i = D_1^j \text{ and } D_2^i < D_2^j).$$

**Definition III:** The scheduling policy EDF2 is the policy that at every time slot selects the configuration with minimum lexicographical deadline vector.

EDF2 is the natural extension of the EDF to the  $2 \times 2$  case. Suppose that a sequence of packets with deadlines are given, and it is known that the deadlines are satisfiable, i.e., there is a feasible scheduling that meet all the deadlines. Then, the scheduling policy EDF2 also satisfies the deadlines. The proof of this claim is straightforward and is similar to the proof of same result for the EDF policy in the single link case.

The EDF2 policy can be applied to obtain a tracking policy. In that case, at every time slot  $k$ , deadline of packets are set to the end of the time slot that they depart the switch under the fluid policy. The departing times are calculated based on the assumption that there is no future arrivals. Note that the crucial information for scheduling is the relative order of packets departure times not the departing times themselves. If we assume that the future arrivals will not change the departure order of the packets, then the departure orders obtained under the no future arrival assumption will be correct, regardless of future arrivals. Therefore, the EDF2 policy would be a non-anticipative tracking policy.

As we will illustrate in the next section, the basic assumption regarding the independence of future arrivals and the departure order of backlogged packets is not a reasonable assumption for the general case of  $N \times N$  switches.

## V. HEURISTIC ALGORITHMS

Let  $\mathcal{F}$  be a feasible fluid policy that at every time slot  $k$  specifies the appropriate fluid scheduling matrix  $w(k)$ . The scheduling matrix  $w(k)$  is an  $N \times N$  matrix indicating the rate of transmission from every input port to every output port, and is a function of arrivals,  $(a_k)$  and backlogged traffic  $(q_k)$  at time  $k$ .

In general, the arrival process is non-anticipative, and therefore the scheduling function is not known in advance. Under these circumstances, it is impossible to track the fluid policy perfectly for  $N > 2$ . We will illustrate this by an example.

- Example: Consider a  $4 \times 4$  switch. Let

$$w(1) = w(2) = \begin{bmatrix} 0.25 & 0.25 & 0.25 & 0.25 \\ 0.25 & 0.25 & 0.25 & 0.25 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}.$$

Without loss of generality, assume that the tracking policy selects the following two sub-permutation matrices for the two

rst time slots.

$$J(1) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, J(2) = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}.$$

Now assume that the uid policy, because of arrival of some packets with higher priority, takes the following rate matrix for the next time slot,

$$w(3) = \begin{bmatrix} 0 & 0.5 & 0 & 0.5 \\ 0.5 & 0 & 0.5 & 0 \\ 0.5 & 0 & 0 & 0.5 \\ 0 & 0.5 & 0.5 & 0 \end{bmatrix}.$$

It is clear that the uid policy finishes four packets on the links (1,2), (1,4), (2,1), (2,3) in the third time slot, and none of them is scheduled in the first two time slots by the packetized policy. Thus, at the end of third time slot, the tracking policy will, at least miss deadlines of 2 packets.

It is worth mentioning that one of the crucial assumptions in the single link sharing case is that the departure order of the packets present in the node is independent of future arrivals. This is not a valid and justifiable assumption in the  $N \times N$  case. The properties of non-anticipative tracking algorithms for single link case is due to this assumption, so one should not expect that similar results regarding the non-anticipative tracking policies hold for  $N \times N$  switches. Recall that we have already provided a non-anticipative tracking policy for the  $2 \times 2$  case. The above example shows that result can not be generalized to the  $N \times N$  case.

This fact motivates us to seek for heuristic algorithms with good but not perfect tracking properties. We are not intending to provide a complete efficient tracking policy here, but we want to illustrate what can be the appropriate approach to design the tracking policies. Basically, we introduce two main concepts that could be employed in the design of tracking policies. We then illustrate the simulation results of a simple tracking policy implemented based on these two concepts.

The two concepts that we elaborate on them in this section are port based tracking and critical links.

### A. Port Based Tracking

One way to implement a packetized tracking policy can be based on the weighted matching in the bipartite graphs. In this method, the weights of the links are the difference between the uid policy and the tracking policy total job done on that link. We call these weights the tracking weights,  $t_{ij}(k)$ .

$$t_{ij}(k) = W_{ij}(k) - I_{ij}(k).$$

We add up all the positive weights of links associated with every input and output port and assign this weight to the corresponding vertex in the bipartite graph. We show the vertex

weights by  $v_i$  and  $v_j$ ,

$$v_i = \sum_{j=1}^N \min(t_{ij}(k), 0), \quad (10)$$

$$v_j = \sum_{i=1}^N \min(t_{ij}(k), 0). \quad (11)$$

If we had added up weight of all links, regardless of their sign, the ones with negative weight would have lessened the total weight of the vertex and this will reduce the chance of those with positive weights (the lagging ones) to be scheduled.

Next, we have to select a criterion function of weights of nodes in the matching, or perhaps, the ones left out of the matching that we attempt to optimize. We propose two possible candidates here.

**Summation Criterion:** One possible candidate is to maximize the summation of all scheduled vertices weights. In this way, at every step we select the set of the nodes with overall maximum weights, and therefore the overall lagging of the tracking policy is minimized.

**Prioritization Criterion:** The other possible choice would be a prioritization criterion. Nodes are prioritized based on their weights, such that the nodes with greater weight have higher priority. Any node is excluded from the matching if and only if including it necessitates removing a node with higher priority from the matching. In this approach the absolute lagging value of a node is considered essential and the ones with greater lagging weight are scheduled.

It is not hard to show that both of the above mentioned criteria are equivalent. This is due to special structure of bipartite graphs. In the next lemma we prove the equivalence of the two criteria.

**Lemma 1:** Let  $M_1$  represents the optimal matching graph based on the summation (priority) criterion. Then,  $M_1$  is optimal based on the priority (summation) criterion too.

**proof:** Suppose that  $M_1$  represents the optimal matching graph based on the summation criterion, but it is not optimal based on the priority criterion. Let  $M_2$  be the optimal matching graph based on the priority criterion. Let  $i_1$  be a node in  $M_1$  but not in  $M_2$ . Consider the graph  $G = M_1 \oplus M_2$  (This graph consists of links that are in one and only one of the two graphs  $M_1$  and  $M_2$ ). The maximum degree of a vertex in  $G$  is two, therefore it consists of a union of distinct paths and loops.  $i_1$  should be the first vertex in a path in  $G$ , because its degree is one. We focus on this path.

Two alternative cases are possible, number of nodes in the path are either even or odd. If it is even the last node in the path is also a member of  $M_1$ , while all intermediate nodes belong to both matching. Thus, if we replace the alternative set of links in the path belonging to  $M_2$  with those belonging to  $M_1$ , two more vertices will be included to  $M_2$ , and this contradicts with the optimality assumption of  $M_2$ .

If number of the vertices in the path is odd, the last vertex is a member of  $M_2$  only. The last node should not have less

weight than the first vertex  $i_1$ , otherwise  $M_2$  is not the optimal graph based on the priority criterion. It should not have greater weight, because this contradicts with the optimality of  $M_1$ . Thus, its weight should be equal to  $i_1$ 's.

Therefore, for every node in  $M_1$  there is a node in  $M_2$  with equal weight and vice versa. This means that both graphs are optimal according to both criteria.

◊

The above argument also provides an algorithm to derive the optimal matching graph. The algorithm is based on exploring the augmenting path similar to the case of maximum matching algorithm. The only difference is that in the maximum matching case the search results a better matching, whenever a free node is detected. Here, the search results a better matching, if a free node or a node with smaller weight than the first node of the augmenting path is detected. The following algorithm finds the maximum weighted vertex matching.

Matching Algorithm:

1. Sort the nodes according to their weights.
2. Select the highest weight node not in the matching that is not selected yet.
3. Search for an augmenting path started from the selected node in step 2. Search ends successfully either if a free vertex is detected (an augmenting path is found), or when a node with smaller weight than the first node is found. Search ends unsuccessfully if all possible paths are searched and none of the above mentioned cases are occurred.
4. Repeat steps 2 and 3 until all nodes are selected.

Although bipartite matching algorithms have been extensively used in scheduling of the switches, they are either based on the maximum link (edge) weighted matching or maximum matching algorithms. The problem with maximum link weighted matching algorithms are their complexity, and the problem with maximum matching algorithms are their poor performance. The vertex based matching can be considered as an intermediate solution.

We can enhance the performance of a scheduling algorithm based on vertex maximum matching by selecting an appropriate initial set of eligible links, which will be provided to the arbiter, and by modifying the weights of the nodes that are more urgent to be scheduled. Basically, we consider a two stage scheduling. At the first step a set of eligible links for scheduling and weights of vertices are derived. In the next step, the arbiter select the links in the matching based on the optimum vertex weighted matching algorithm described above.

One way to select an eligible set of links in the tracking problem is based on the notion of critical links. The critical links are those that are urgent to be scheduled, and if they are not scheduled the tracking policy will loose the track of the fluid policy. If such a link is detected all non-critical links sharing same input or output port with the critical link will be excluded. The precise definition of critical nodes is given in the next section.

## B. Critical Ports and Links

A critical port is a port that should be scheduled in the next time slot, in order not to miss a deadline in the future. As an example suppose that we are at the beginning of  $k - th$  time slot. Let say that there are two packets one from node  $i$  to  $j_1$  and the other from node  $i$  to  $j_2$  both with deadline  $k + 1$ . Note that if we do not schedule any of these packets, no deadline will be missed in the  $k - th$  time slot. However, we will definitely miss a deadline at the subsequent time slot,  $k + 1$ . We say that node  $i$  is a critical node, and links  $(i, j_1)$  and  $(i, j_2)$  are associated critical links. In general, sufficient condition for a port to be critical at time  $k$  is to have at least  $p$  packets with deadlines less than or equal to  $k + p$ . Denote that we are stating a sufficient condition. In other words, there might be some critical ports that can not be detected well in advance using this criterion.

In case of the tracking policy, the deadline of packets are implicitly given, and are equal to the end of the time slot that the packet departs the switch under fluid policy. We may not know the deadlines well in advance, since the future rate of every link under the fluid policy depends on the future arrivals, which are non-anticipative. Nevertheless, we may have an approximate deadline for every packet based on back-logged traffic or the average arrival rate of the links (for instance, based on a  $(\sigma, \rho)$  flow model for ATM traffic). Also, in some approaches, a constant rate is assigned to a session, regardless of the arrival process, such as in multi-periodic messages in TDMA-SS or rate controlled service disciplines. In the case of networks if no packet from the assigned session is available at time of scheduling, packets from other kind of traffic (for instance, the best effort traffic) can use the available slot [14].

The next issue is to set an appropriate inspection horizon. The inspection horizon is the number of time slots in future that we inspect to detect the critical nodes. Based on our experiments, we found that inspection horizon around five is adequate. In fact, there is a trade-off involved here, increasing the inspection horizon helps us in detecting more critical links, but it increases the complexity of the algorithm as well.

After detecting a critical port, we know that we have to schedule one of the critical links associated with that, otherwise we will miss the deadlines. Thus, we remove all links associated with critical port, which are not critical. In this way, the chance for the scheduler to arbitrate one of the critical links is increased. We also increase the weight of the critical ports with a constant, so that their weight become greater than all non-critical ports. Therefore, these nodes are prioritized by the scheduler.

The critical nodes detecting algorithm can be described as follow,

Critical Node Detecting Algorithm:

1. For every node  $i$  and  $j$  do steps 2 and 3;
2. Calculate number of Packets that should be sent in the next  $p$  time slots.

3. If the number of packets that should be sent in the next  $l$  time steps is equal to  $l$ , the corresponding node is critical. Moreover, the associated links that should at least send one packet in the next  $l$  time slots are critical links.

The whole scheduling process of a switch can be divided into two stages. In the first stage, weight of the ports are calculated and the criticality of the ports are investigated. These computation can be done in parallel for different ports of the switch, and no interaction between them is necessary. In the next stage the computed weights for the nodes and the eligible links for every node are provided to the arbiter processor.

One of the main concerns in high speed switches is the volume of information required to be exchanged between different cards of the switch, namely the signaling scalability of the algorithm. In the link weighted matching arbiters, weights associated to every link should be sent to the arbiter. Thus, the exchanging information is in the order of  $N^2$ . In our approach the weights of the nodes and the eligible links (one bit information per link) should be sent to the arbiter, which is in the order of  $N$ .

Finally we provide a simple scheduling algorithm based on the algorithms described above.

Heuristic Scheduling Algorithm:

1. At every time slot  $k$  do the following steps.
2. Calculate weights of the nodes using (10),(11).
3. Insert all links with positive weights in eligible links set.
4. Check for critical nodes and their associated critical links.
5. Increase weight of every critical node, such that its weight exceed all non-critical node's weights. Moreover, remove all non-critical links of the critical nodes from the eligible links set.
6. Pass weights of the nodes and the critical links set to the matching algorithm. The result of the matching is the schedule for time slot  $k$ .

This algorithm will be used in next section for scheduling in a TDMA-SS, and the simulation results will be illustrated.

### C. Simulation Results

In this section we provide some primary results regarding the heuristic algorithm. The thorough investigation and analysis of the heuristic algorithms are still continuing. However, the primary results appear promising. We have used the heuristic algorithm for scheduling of a TDMA-SS with multi-periodic messages. Suppose that messages are all periodic, and the objective is to schedule every period of a message before the next one arrives. Thus, if the period of message  $M_i$ , is  $p_i$ , we have  $p_i$  time slots to schedule every packet of that message. The fluid policy that assigns the constant rate of  $1/p_i$  to message  $M_i$  accomplishes this task.

The messages are generated randomly between input and output pairs with equal probability. The period of the message is selected uniformly in the range of  $\{3, \dots, 8\}$ . The messages are selected such that in every experiment the utility of every node is in  $[0.90, 0.97]$ . In all experiments the average utility is

TABLE I  
HEURISTIC ALGORITHM PERFORMANCE (LATENCY) FOR DIFFERENT SWITCH SIZES

N	utility	l=0	l=1	l=2	late %
5	0.92	459699	293	2	0.064
10	0.92	922195	390	8	0.043
20	0.92	1843935	559	8	0.031
30	0.92	2770131	684	9	0.025
50	0.92	4620472	884	8	0.019
70	0.92	6470463	1032	11	0.016
100	0.92	9191925	1209	12	0.013

around 0.92. The critical nodes are inspected based on the constant rate, and the inspection horizon is set to  $N$ . We have done the simulation for different switch sizes, and for every switch size we generated 100 different message sets and investigate the performance of the algorithm.

In our model there can be different messages from the same input to the same output, but the scheduler works with the aggregate rate of all these messages. This is important, because the complexity of the arbiter does not depend on the number of messages, which can be large. To schedule different messages from the same input to output an EDF scheduler is maintained in every input port. A packet is considered to be on time (zero latency) if it is scheduled within a period interval after its arrival. The latency of a packet,  $l$  is equal to  $k > 0$  if it is scheduled  $k$  time units after its deadline. The result of the simulations are given in the table below. As far as the scheduler is concerned, it is working with constant rate traf cs. Therefore, the results of this simulation are also applicable in the case of scheduled rate scheduler for the network switches

The number of packets with different latencies are indicated in the table. The percentage of late packets is also shown in the last column. Every row correspond to a different switch size. The number of packets missing their deadlines are less than one percent, and the maximum latency is two time units in all cases. We believe that in many of the applications this is an acceptable performance. In many applications the excessive delay imposed by the arbiter is tolerable. In fact, in some of the applications we are allowed to miss some packets, so we can neglect the packets that miss their deadline. This in return aids us in on time scheduling of the other packets. The other important issue is the size of the switch. Many of the heuristic algorithms proposed degrades as the size of the switch increases [13]. In our case, we did not observe any such degradation.

## VI. SUMMARY AND CONCLUSION

In this paper the notion of fluid policies and tracking policies are extended to the  $N \times N$  switches. These concepts are both useful in the high speed networks, where they aid us in providing guaranteed service to different applications, and in TDMA-SS with multi-periodic messages. The existence of tracking

policy is proved for the special case of  $2 \times 2$  switches. For the general case of  $N \times N$  switches a heuristic algorithm is provided. The heuristic algorithm is mainly presented to confirm the validity of two important notions and measures in tracking policies, the port tracking and critical node concepts.

The existence of tracking policy for the  $N \times N$  is still an open question. However, based on the results provided here for the special cases and the heuristic algorithm, we think that such a tracking policy always exist. However, we have shown that it is impossible to have a perfect tracking policy when the fluid policy is non-anticipative. This fact together with the complexity issue justify the need for better and less complicated heuristic tracking policies, with good but not perfect performances.

#### REFERENCES

- [1] J.C.R. Benett and H. Zang. WF2Q: Worst-case Fair Weighted Fair Queueing. Proceedings of INFOCOM '96, IEEE, March 1996.
- [2] S. Chuang, A. Goel, N. McKeown, B. Prabhakar. Matching output queueing with a combined input output queued switch. Proceedings of INFOCOM '99, 1169-1178, IEEE, April 1999.
- [3] R. Cruz. A calculus for network delay. I. Network elements in isolation. IEEE Trans. on Information Theory, 37(1):114-131, January 1991.
- [4] R. Cruz. A calculus for network delay. II. Network Analysis. IEEE Trans. on Information Theory, 37(1):132-141, January 1991.
- [5] A. Demers, S. Keshav, and S. Shenkar. Analysis and simulation of a fair queueing algorithm. Proceedings of SIGCOM '89, pages 1-12.
- [6] L. Georgiadis, R. Guerin, and A. Parekh. Optimal multiplexing on single link: Delay and buffer requirements. pages 524-532, 1994.
- [7] L. Georgiadis, R. Guerin, V. Peris, K.N. Sivarajan. Efficient network QoS provisioning based on per node traffic shaping. IEEE/ACM Transactions on Networking, vol.4, no.4, pp. 482-501, 1996.
- [8] J. Giles, B. Hajek. Scheduling multirate periodic traffic in a packet switch Shaping. Conf. on Info. Sci. and Systems at John Hopkins Univ, 1997.
- [9] S. Golestani. A self-clocked fair queueing scheme for broadband applications. Proceedings of INFOCOM '94, 636-646, IEEE, April 1994.
- [10] T. Inukai. An efficient SS/TDMA time slot assignment algorithm. IEEE Transactions on Communications, vol.27, pp. 1449-1455, 1979.
- [11] A.K. Parekh and R.G. Gallager. A generalized processor sharing approach to flow control in integrated services networks: The single node case. IEEE/ACM Transactions on Networking, 1(3):344-357, June 1993.
- [12] A.K. Parekh and R.G. Gallager. A generalized processor sharing approach to flow control in integrated services networks: The multiple node case. IEEE/ACM Transactions on Networking, 2(2):137-150, April 1994.
- [13] I.R. Philip. Scheduling real-time messages in packet-switched networks. Ph.D. Dissertation Dept. Comp. Sci. Univ of Illinois at Urbana-Champaign, 1997.
- [14] H. Zhang, D. Ferrari. Rate-controlled service disciplines. J. High Speed Networks, vol.3, no.4, pp. 389-412, 1994.