QoS Provisioning and Tracking Fluid Policies in Input Queueing Switches

Vahid Tabatabaee[1], Leonidas Georgiadis[2], Leandros Tassiulas[1]
[1] Department of Electrical and Computer Engineering and Institute for Systems Research
University of Maryland at College Park
[2]Electrical and Computer Engineering Department
Aristotle University, Thessaloniki, Greece

Abstract

The concept of tracking fluid policies by packetized policies is extended to input queueing switches. It is considered that the speed up of the switch is one. One of the interesting applications of the tracking policy in TDMA satellite Switches is elaborated. For the special case of 2×2 switches it is shown that a tracking non-anticipative policy always exists. It is found that in general non-anticipative policies do not exist for switches with more than 2 input and output ports. For the general case of $N \times N$ witches a heuristic tracking policy is provided. The heuristic algorithm is based on two notions, port tracking and critical links. These notions can be employed in the derivation of other heuristic tracking policies as well. Simulation results show 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 services networks is to provide performance requirements to a broad range of applications. Application 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, [6],[7],[15],[16],[9]. 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 a 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 [15],[16],[9],[8],[12]. 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 [5] 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 2×2 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.

The heuristic algorithm is based on a weigthed matching algorithm. The weighted matching algorithms are usually too complex to implement in hardware. Here we employ two notions to reduce the complexity. The first concept is to do port based weighted matching rather that link weighted matching. Mekkittikul and McKeown [14] used a similar concept. They used the queue length as the weights in their work, and illustrate that they can achieve 100% throughput. Our weights reflect the amount of work by which the fluid policy is ahead of the tracking policy at each port, and our main objective is to be able to provide rate guarantees and high throughput at the same time. The second notion that aids us in improving the performance and reducing the complexity is the concept of critical ports and links. Basically, criticality relates to the urgency by which a port/link needs to be scheduled in order to ensure proper tracking. We detect all critical links and remove all non-critical links that conflict with the critical ones. In this way the number of contending links and consequently the complexity of the algorithm is reduced.

An interesting application of 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 tracks that fluid policy.

The organization of paper is as follows. In the next section, we 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. It is indicated that this problem is essentially a special case of the input queueing scheduling problem considered in this paper. In section IV, we show that for the 2×2 switches a tracking policy always exist, and we provide a non-anticipative algorithm to find the tracking policy. In section V, some useful ideas regarding the design of heuristic tracking policies are given. Based on these concepts a heuristic scheduling algorithm is proposed, and used to implement a fixed rate scheduler.

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.

Two broad classes of policies are considered, fluid and 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.

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

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 input port i to output 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,

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

Note that the conditions in (2) imply that for any k, there can be at most a single 1 in each column and 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, as mentioned in the introduction, 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 π_f, we say that a packetized policy is *tracking* π_f 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 [15], [9]. In that case, perhaps the best known fluid policy is the Generalized Processor Sharing (GPS) policy. Several tracking policies are suggested for the single link case [15], [9], [12]. The concept of GPS can be appropriately extended to the multi-input,multi-output input queueing switches. However, the existence of tracking policies for these switches is still an open question. In section IV, we study the special case of 2×2 switches. We will prove that for the special case of 2×2 switches, for every feasible fluid policy there exists a feasible packetized policy. In fact, our proof is constructive and provides an algorithm to derive a tracking policy. Before discussing the 2×2 case, in the next section we present the problem of multi-periodic TDMA-SS scheduling and indicate how the problem could be trivially solved by the construction of tracking packetized policies.

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 [13]. 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 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 have period p_m . To apply the Inukai method the frame length should be set to the Least Common Multiplier (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.,

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

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 fix 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. In [17] Philp and Liu conjectured that (3) is the necessary and sufficient condition for schedulability under the specified delay constraints. Giles and Hajek [11] have proved this conjecture for a special case. In their model the messages are sorted based on their period, such that,

$$p_1 \geq p_2 \geq \cdots \geq p_M.$$

Moreover, for every two subsequent messages we have,

$$p_m = k p_{m+1},$$

where k is an integer. Unfortunately, their algorithm does not work well in the general case. More recently, Bonuccelli and Clo [3] present a counter-example for a 4×4 switch, and illustrate that the conjecture is in general not correct. In the next section, we show the existence of tracking policies for the special case of 2×2 switches. Thus, the conjecture is proved for the special case of 2×2 switches.

IV. The 2×2 switch

In this section, we consider a 2×2 input queueing switch and provide an algorithm for designing a packetized policy π that tracks a given fluid policy π_f.

We make the following assumption regarding the fluid policy.

Assumption I: The fluid policy is non-anticipative (its decisions do not depend on future arrivals) and such that the order in which packets with origin port i and destination port j complete service, is not affected by new packet arrivals.

This assumption is similar to the one used in the design of tracking fluid policies in the single server case [9]. Note that the assumption does not preclude the possibility that new packet arrivals affect the order in which some packets complete service. Consider the following example of a scheduling policy in a 2×2 switch (see Figure 1). Input port 1 employs a GPS scheduler to schedule packets destined to output ports $1, 2$ with weights $1/4$ and $3/4$ respectively. The scheduler operates in a work-conserving fashion, serving all eligible packets (i.e., packets that have no transmission constraints at any of the output ports) according to their weights. Output port 2 uses a strict priority scheme to schedule packets from input ports 1 and 2; packets from input port 2 have higher priority. Assume now that at the beginning of time slot 1, two packets p_1, p_2 arrive at input port 1, destined to output ports 1 and 2 respectively. If no new arrival occurs at the beginning of slot 2, then the finishing times of p_1 and p_2 are 2 and $4/3$ respectively, as shown in Figure 1, a). Assume next that at the beginning of time slot 2, packet p_3 arrives at input port 2, destined for output port 2 -see Figure 1, b). Then at time slot 2, packet p_3 will be transferred from input port 1 to output port 2, since at

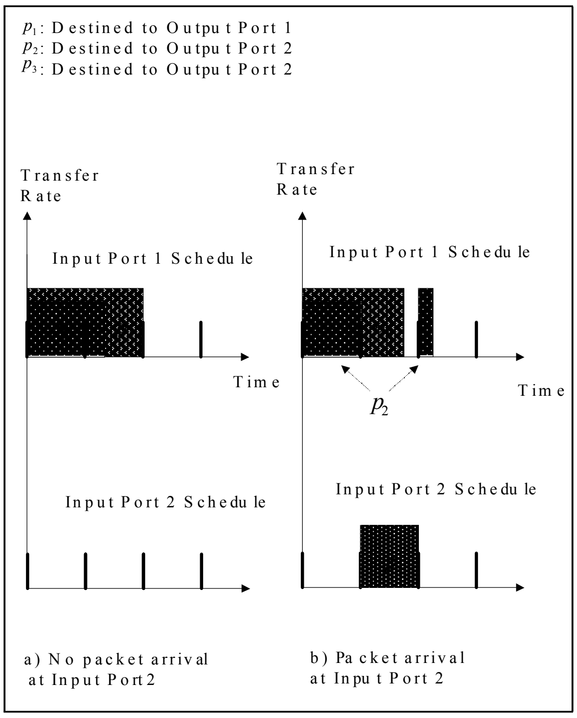
p_1: Destined to Output Port 1
p_2: Destined to Output Port 2
p_3: Destined to Output Port 2
Transfer Rate
Input Port 1 Schedule
Time
Input Port 2 Schedule
a) No packet arrival at Input Port2
Transfer Rate
Input Port 1 Schedule
p_2
Time
Input Port 2 Schedule
b) Packet arrival at Input Port 2

Fig. 1. Effect of packet p_3 on completion times of p_1 and p_2.

output port 2 packet p_3 has higher priority than packet p_2. Since p_2 cannot be transmitted in slot 2, the GPS scheduler at input port 1 completes transmission of p_1 by time 7/4. The completion time of packet p_2 is now 9/4. We see that the order by which packets p_1 and p_2 complete service is reversed by the arrival of packet p_3. If, however, packets destined to the same input-output port pair (i, j) are served according to, for example, a strict priority scheme, then the specified switch scheduling policy satisfies Assumption I.

Examples of fluid policies that satisfy Assumption I are:

• Any nonanticipative fluid policy that serves packets with origin port i and destination port j in a First Come First Served (FCFS) manner.

• Any fluid policy that assigns fixed priorities to packets with origin port i and destination port j

• A policy that employs a general non-anticipative fluid scheduler at input port i to provide transmission intervals to packets destined to different output ports, and another GPS scheduler to schedule, within the provided transmission intervals, packets at port i destined to a particular output port j.

The main difficulty in the design of tracking policies for input switches arises from the fact that if one sees an input-output port (i, j) as a server, this server will not be work-conserving, since it may be forced to idle at some slot k. This can happen if, for example, during slot k a packet is transmitted from input port i to another output port j_1, instead of the packet destined to output port j. Below we approach the design of the tracking policy in two steps: First we design a sequence of sub-permutation matrices $I_{ij}(k)$ that "tracks" the work performed by the fluid policy for any origin-destination pair. The meaning of tracking in this case is provided below in (4). The interpretation of the matrix $I_{ij}(k)$ is the following. Whenever $I_{ij}(k) = 1$, a packet with origin i and destination j may be transferred through the switch at slot k (if there is such a packet in the queue). Next, we provide a packetized policy, π_p, that decides which packets among those using a particular origin-destination pair are to be transmitted at the slots specified by $I_{ij}(k)$.

Before we proceed we need the following definition.

Definition II: $\ell_{ij}(k)$ is the largest time slot less than or equal to k, at the beginning of which there is no work at port i destined for port j under π_f.

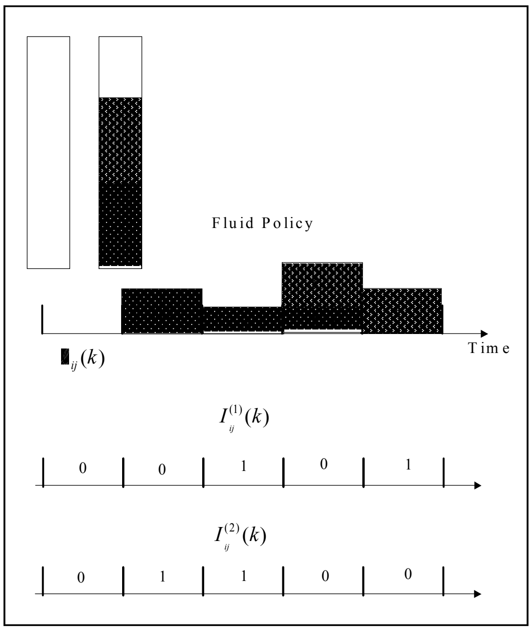

Fig. 2. Packet transfer from input port i to output port j.

s	$\ell_{ij}(k)$	$\ell_{ij}(k)+1$	$\ell_{ij}(k)+1$	$\ell_{ij}(k)+3$	$\ell_{ij}(k)+4$
$w_{ij}(s)$	0	1/2	1/4	3/4	1/2
$\sum_{s=l}^{k} w_{ij}(s)$	2	2	3/2	5/4	1/2
$\left\lfloor \sum_{s=l}^{k} w_{ij}(s) \right\rfloor$	2	2	1	1	0
$\sum_{s=l}^{k} I^{(1)}_{ij}(s)$	2	2	2	1	1
$\sum_{s=l}^{k} I^{(2)}_{ij}(s)$	2	2	1	0	0

TABLE I
Example of Sub-mermutation Matrices

Let $I_{ij}(k)$ be a sequence of integer sub-permutation matrices that have the following property.

$$\max_{\ell_{ij}(k)\leq l\leq k}\left\{\left\lfloor\sum_{s=l}^{k} w_{ij}(s)\right\rfloor-\sum_{s=l}^{k} I_{ij}(s)\right\}\leq 0,\ k=1,2,\ldots \tag{4}$$

where $\lfloor x\rfloor$ denotes the integer part of x. The design of $I_{ij}(k)$ will be provided later in this section. Consider the example in Figure where two sequences, $I^{(1)}_{ij}(k)$ and $I^{(2)}_{ij}(k)$ are provided. From Table I we see that the sequence $I^{(1)}_{ij}(k)$ satisfies (4), while $I^{(2)}_{ij}(k)$ does not, since $\left\lfloor\sum_{s=\ell_{ij}(k)+3}^{k} w_{ij}(s)\right\rfloor - \sum_{s=\ell_{ij}(k)+3}^{k} I_{ij}(s) = 1$.

Note that $I_{ij}(k)$ specifies whether in a slot there may be a transfer of a packet between ports i and j (if $I_{ij}(k) = 1$ and there is work at input port i at time $k-1$ destined for output port j), but it does not specify which packet from the corresponding queue will be chosen for transfer. We now define a packetized policy π_p, that specifies the packet to be chosen for transfer in such a way that the fluid policy π_f is tracked.

Definition III: π_p is the packetized policy that whenever $I_{ij}(k) = 1$ and there is work at input port i at time $k-1$ destined for output port j, it transfers the packet that completes earliest under π_f.

Provided that $I_{ij}(k)$ can be designed in a non-anticipative fashion, π_p is also non-anticipative, since by Assumption I, one can decide the order of completion times of packets with the same origin-destination pair without knowledge of future arrivals.

Note that π_p acts in the same manner that the tracking policy in the single server case acts [9], except that it first checks, based on $I_{ij}(k)$, whether a slot is eligible for transmission of packets with origin-destination pair (i,j). On the other hand, the specified policy can be considered as a generalization of the policy for the single-server case. Indeed, specializing to the single-server case, we define $I_{ij}(k)=1$ whenever there is backlog in the system at time $k-1$ under π_f and $I_{ij}(k)=0$ otherwise. Then (4) is obviously satisfied, and the resulting policy is identical to the tracking policy for the single server case, proposed in [9].

The next theorem shows that π_p is tracking π_f. In the following it is assumed that the system is empty at time 0.

Theorem 1: Every packet leaves the switch under π_p at the latest by the end of the time slot at which the same packet leaves the switch under π_f.

Proof: Let b_m be the mth time slot at the beginning of which there is no work at input port i destined for output port j under π_f, while at the end of b_m there is such work. In other words, b_m is the beginning of the mth busy period under π_f, for the server related to the pair (i,j). Let also e_m be the end of the mth busy period under π_f, that is, the first time slot after b_m at the beginning of which there is work at input port i destined for output port j under π_f, while at the end of e_m there is no such work. The theorem is true until time slot b_1. Assume that the theorem is true for all packets that are transmitted up to time slot b_m under π_f. Notice that then, according to the statement of the theorem it follows that the same packets have been transmitted up to time slot b_m under π_p.

Next we will show that the theorem holds for all packets that arrive and are transmitted from time b_m to e_m, and therefore, the theorem holds up to time slot b_{m+1}. Let p_n be the nth packet with origin port i and destination port j to complete transmission in the interval $(b_m,e_m]$ under π_p. Let f_n ($\widehat{f_n}$) be the finishing time of p_n under π_p (under π_f). We will show that

$$f_n \le \left\lceil \widehat{f_n} \right\rceil$$

($\lceil x \rceil$ denotes the smallest integer larger or equal than x). For the proof, assuming the contrary, i.e.,

$$f_n > \left\lceil \widehat{f_n} \right\rceil, \text{ or } f_n - 1 \ge \left\lceil \widehat{f_n} \right\rceil, \tag{5}$$

we will arrive at a contradiction.

Consider two cases.

Case 1. Suppose that for all packets p_l, $l<n$, it holds (see Figure 3)

$$\widehat{f_l} \le \widehat{f_n}.$$

Then, packets $p_1, p_2, ..., p_{n-1}$ leave before packet p_n under both π_p and π_f. Let $\overline{\ell} \ge b_m$ be the last slot before f_n such that $I_{ij}(\overline{\ell})=1$ and there is no packet to be transferred from port i to port j under π_p at time $\overline{\ell}-1$. If there is no such slot, set $\overline{\ell}=b_m$. Let also p_l, p_{l+1},...,p_n be the packets that are transmitted in the interval $[\overline{\ell},\ f_n]$ under π_p. Note that all these packets must have arrived at or after time $\overline{\ell}$. This is so since either $\overline{\ell}=b_m$ and the statement is true by definition, or $I_{ij}(\overline{\ell})=1$, and therefore, if one of these packets was in the system at the beginning of slot $\overline{\ell}$, it would have been transmitted in that slot. Therefore, we have the following important properties

- Packets p_l, p_{l+1},...,p_n are transmitted in slots $\overline{\ell}+1$ to f_n under π_p and in slot $\overline{\ell}+1$ to $\widehat{f_n}$ under π_f.
- Whenever $I_{ij}(k)=1$, $\overline{\ell}+1 \le k \le f_n$, one of the packets p_l, p_{l+1},...,p_n is transferred from port i to port j under π_p.

Therefore,

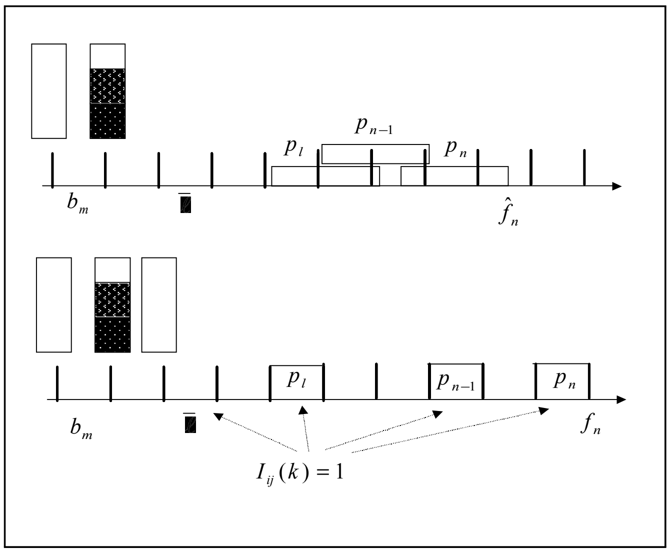

Fig. 3. Arrangement of packets for Case 1 in the proof of Theorem 1

$$\sum_{s=\bar{\ell}}^{\lceil \widehat{f_n} \rceil} w_{ij}(s) \geq \sum_{s=\bar{\ell}}^{f_n} I_{ij}(s) \tag{6a}$$

$$= \sum_{s=\bar{\ell}}^{f_n-1} I_{ij}(s) + 1 \tag{6b}$$

$$\geq \sum_{s=\bar{\ell}}^{\lceil \widehat{f_n} \rceil} I_{ij}(s) + 1 \tag{6c}$$

$$\geq \left\lfloor \sum_{s=\bar{\ell}}^{\lceil \widehat{f_n} \rceil} w_{ij}(s) \right\rfloor + 1 \tag{6d}$$

Equality (6b) follows from the fact that since p_n was transmitted in slot f_n, $I_{ij}(f_n) = 1$, inequality (6c) follows from (5), and inequality (6d) from (4). Finally, we have the contradiction,

$$\sum_{s=\bar{\ell}+1}^{\lceil \widehat{f_n} \rceil} w_{ij}(s) \geq \left\lfloor \sum_{s=\bar{\ell}+1}^{\lceil \widehat{f_n} \rceil} w_{ij}(s) \right\rfloor + 1$$

Case 2. Suppose there is a packet p_l, $l < n$ such that

$$\widehat{f_n} < \widehat{f_l}$$

i.e., packet p_l leaves earlier that packet p_n under π_p and later than packet p_n under π_f (see Figure 4). Assume that p_l is the last packet before p_n with this property. Then, packets $p_{l+1,\ldots,}p_n$ leave earlier than, or at the same time as, packet p_n under both policies. This implies that packets $p_{l+1},\ldots,p_n$ must have arrived at the earliest at the end of slot f_l. This is so, since otherwise, according to the definition of π_f and Assumption I, if one of the packets $p_{l+1},\ldots,p_n$ was in the system at the beginning of slot f_l, it would have been transmitted earlier than p_l since its finishing time under π_f is smaller than the finishing time of $p_{l.}$ Therefore, we conclude that packets $p_{l+1},\ldots,p_n$ arrive and are transmitted in the interval $[f_l, f_n]$ under π_p and in the interval $[f_l, \widehat{f_n}]$ under π_f. Using again the argument in case 1, we arrive at a contradiction.

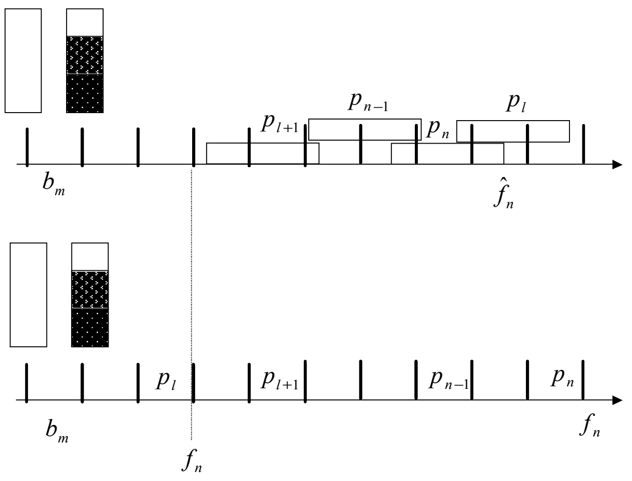

Fig. 4. Packet arrangement for Case 2 in the proof of Theorem 1

■

It remains to show that the matrix $I_{ij}(k)$ can be constructed in a non-anticipative manner, based on π_f. We will in fact construct a policy that in addition to (4) has the following property.

Property I. The following inequalities hold for all k.

$$\sum_{j=1}^{2} w_{ij}(k) \le \sum_{j=1}^{2} I_{ij}(k),\ i=1,2;\ \sum_{i=1}^{2} w_{ij}(k) \le \sum_{i=1}^{2} I_{ij}(k),\ j=1,2.$$

Define next $I_{ij}(k)$ recursively as follows

$$I_{ij}(k+1) = \max_{\ell_{ij}(k+1)\le l\le k+1} \left\{ \left\lfloor \sum_{s=l}^{k+1} w_{ij}(s) \right\rfloor - \sum_{s=l}^{k} I_{ij}(s),\ 0 \right\},\ k \ge 0, \tag{7}$$

where the notation $\sum_n^m = 0$ when $n > m$, is used. Note that since π_f is non-anticipative, $w_{ij}(k+1)$ can be computed at time k based only on the past history of the system. Since $w_{ij}(s)$, $I_{ij}(s)$, $0 \le s \le k$ are also known at time k, $I_{ij}(k+1)$ can indeed be computed at time k. In the next Lemma we show that the matrix $I_{ij}(k+1)$ computed using (7), with a slight modification, satisfies (4).

Lemma 2: At slot $k+1$,compute $I_{ij}(k+1)$ using (7). If it turns out that some row or column of $I_{ij}(k+1)$ contains only zeros, then one of the elements of this row or column can be redefined to one, so that the resulting matrix remains a sub-permutation matrix. The matrix $I_{ij}(k+1)$ so defined, satisfies (4) and property I.

Proof: Assume $I_{ij}(k)$ satisfies (4) and Property I up to slot k. We show now that the same holds for slot $k+1$. First, we have to show that $I_{ij}(k+1)$ is an integer sub-permutation matrix, i.e., $I_{ij}(k+1)$ takes values 0 or 1, and

$$\sum_{j=1}^{2} I_{ij}(k+1) \le 1,\ i=1,2,\ \sum_{i=1}^{2} I_{ij}(k+1) \le 1,\ j=1,2. \tag{8}$$

To show that $I_{ij}(k+1)$ takes values 0 or 1, notice that

$$\begin{aligned}
0 &\le I_{ij}(k+1) && \text{(9a)}\\
&\le \max_{\ell_{ij}(k+1)\le l\le k+1}\left\{\left\lfloor 1+\sum_{s=l}^{k} w_{ij}(s)\right\rfloor - \sum_{s=l}^{k} I_{ij}(s),\ 0\right\} && \text{(9b)}\\
&= \max_{\ell_{ij}(k+1)\le l\le k+1}\left\{1+\left\lfloor \sum_{s=l}^{k} w_{ij}(s)\right\rfloor - \sum_{s=l}^{k} I_{ij}(s),\ 0\right\}. && \text{(9c)}
\end{aligned}$$

Inequality (9b) follows from the fact that $w_{ij}(s) \le 1$. Suppose $\ell_{ij}(k+1) = \ell_{ij}(k)$. Then from (4) we conclude that

$$\left\lfloor \sum_{s=l}^{k} w_{ij}(s)\right\rfloor - \sum_{s=l}^{k} I_{ij}(s) \le 0,\ \ell_{ij}(k+1) \le l \le k$$

and hence

$$\max_{\ell_{ij}(k+1)\le l\le k+1}\left\{1+\left\lfloor \sum_{s=l}^{k} w_{ij}(s)\right\rfloor - \sum_{s=l}^{k} I_{ij}(s),\ 0\right\} \le 1, \tag{10}$$

Taking into account(9c) and the fact that $I_{ij}(k+1)$ is integer-valued, we conclude that $I_{ij}(k+1)$ takes the values 0 or 1.Suppose now that $\ell_{ij}(k+1) \ne \ell_{ij}(k)$. Then by the definition of $\ell_{ij}(k)$, we conclude that $\ell_{ij}(k+1) = k+1$ and $w_{ij}(k+1) = 0$, hence it follows from (7) that $I_{ij}(k+1) = 0$.

In order to show (8) assume that for, say, row 1 we have

$$I_{11}(k+1) = I_{12}(k+1) = 1.$$

Then, be the definition (7) there must be $\ell_1 \ge \ell_{11}(k+1)$ and $\ell_2 \ge \ell_{12}(k+1)$ such that

$$\left\lfloor \sum_{s=\ell_1}^{k+1} w_{11}(s)\right\rfloor - \sum_{s=\ell_1}^{k} I_{11}(s) = 1$$

$$\left\lfloor \sum_{s=\ell_2}^{k+1} w_{12}(s)\right\rfloor - \sum_{s=\ell_2}^{k} I_{12}(s) = 1$$

Assume without loss of generality that $\ell_1 \ge \ell_2$. Then, adding the previous equations, we have

$$\begin{aligned}
2 &\le \left\lfloor \sum_{j=1}^{2} w_{1j}(k+1) + \sum_{s=\ell_1}^{k}\sum_{j=1}^{2} w_{1j}(s) + \sum_{s=\ell_2}^{\ell_1-1} w_{12}(s)\right\rfloor && \text{(11a)}\\
&\quad - \sum_{s=\ell_1}^{k}\sum_{j=1}^{2} I_{1j}(s) - \sum_{s=\ell_2}^{\ell_1-1} I_{12}(s)\\
&\le 1 + \left\lfloor \sum_{s=\ell_1}^{k}\sum_{j=1}^{2} w_{1j}(s) + \sum_{s=\ell_2}^{\ell_1-1} w_{12}(s)\right\rfloor && \text{(11b)}\\
&\quad - \sum_{s=\ell_1}^{k}\sum_{j=1}^{2} I_{1j}(s) - \sum_{s=\ell_2}^{\ell_1-1} I_{12}(s).
\end{aligned}$$

Inequality (11a) follows from the fact that $\lfloor x\rfloor + \lfloor y\rfloor \le \lfloor x+y\rfloor$ and inequality (11b) from the fact that $\sum_{j=1}^{2} w_{ij}(k+1) \le 1$. Hence, taking also into account that $I_{ij}(s)$ are integers, we have

$$1 \le \left\lfloor \sum_{s=\ell_1}^{k}\sum_{j=1}^{2} w_{1j}(s) - \sum_{s=\ell_1}^{k}\sum_{j=1}^{2} I_{1j}(s) + \sum_{s=\ell_2}^{\ell_1-1} w_{12}(s) - \sum_{s=\ell_2}^{\ell_1-1} I_{12}(s)\right\rfloor.$$

Using Property I we conclude that

$$\sum_{s=\ell_1}^{k}\sum_{j=1}^{2} w_{1j}(s) - \sum_{s=\ell_1}^{k}\sum_{j=1}^{2} I_{1j}(s) \le 0$$

and hence

$$1 \le \left\lfloor \sum_{s=\ell_2}^{\ell_1-1} w_{12}(s) - \sum_{s=\ell_2}^{\ell_1-1} I_{12}(s) \right\rfloor \tag{12}$$

If $\ell_2 = \ell_1$ then $\left\lfloor \sum_{s=\ell_2}^{\ell_1-1} w_{12}(s) \right\rfloor - \sum_{s=\ell_2}^{\ell_1-1} I_{12}(s) = 0$, which contradicts (12). If $\ell_2 < \ell_1$, then necessarily $\ell_{12}(k{+}1) = \ell_{12}(k) \le \ell_2 < \ell_1$, and since $I_{ij}(s)$ satisfies (4) for $s \le k$, we have $\left\lfloor \sum_{s=\ell_2}^{\ell_1-1} w_{12}(s) \right\rfloor - \sum_{s=\ell_2}^{\ell_1-1} I_{12}(s) \le 0$, which again contradicts (12).

We also need to ensure that $I_{ij}(k+1)$ satisfies Property I. If $I_{ij}(k+1) = 1$, then clearly Property I holds for column j and row i. Assume next that for a column or a row, say column 1, it is computed based on (7) that

$$I_{11}(k+1) = I_{21}(k+1) = 0.$$

In order to ensure that Property I holds for $k+1$, we claim that we can redefine one of $I_{11}(k+1)$, $I_{21}(k+1)$ to 1 without affecting the sub-modularity property of the matrix. To see this, notice that since $I_{ij}(k+1)$ takes values 0 or 1 and satisfies (8), in column 2 there can be at most a single 1, say in position (1,2). We can therefore set $I_{21}(k+1) = 1$ in order to ensure that Property I holds for column 1. Proceeding in this way, we can redefine some of the $I_{ij}(k+1)$ if necessary, in order to ensure that Property I holds. Notice also that with this redefinition, the resulting matrix $I_{ij}(k+1)$ still satisfies (4).

It remains to show that $I_{ij}(s)$ satisfies (4) for $s = k+1$. Assume first that $I_{ij}(k+1)$ has not been redefined. Since by the definition (7),

$$I_{ij}(k+1) \ge \left\lfloor \sum_{s=l}^{k+1} w_{ij}(s) \right\rfloor - \sum_{s=l}^{k} I_{ij}(s),\ \ \ell_{ij}(k+1) \le l \le k+1,$$

we easily conclude that

$$\max_{\ell_{ij}(k+1)\le l \le k+1} \left\{ \left\lfloor \sum_{s=l}^{k+1} w_{ij}(s) \right\rfloor - \sum_{s=l}^{k+1} I_{ij}(s) \right\} \le 0$$

If any of the $I_{ij}(k+1)$ needs to be redefined, this redefinition only increases the value of $I_{ij}(k+1)$ and hence (4) still holds. ■

V. Heuristic Algorithms

Let π_f be a feasible fluid policy that at every time slot k specifies the appropriate fluid scheduling matrix $w(k)$. We showed in the previous section that as long as π_f satisfies Assumption 1, a non-anticipative tracking packetized policy can be designed for a 2×2 switch. For the general case of $N \times N$ switches the tracking policy does not always exist. Bonuccelli and Claudia present an example [3] for 4×4 switches with fixed rate. This example shows that even an anticipative tracking policy may not exist for $N \ge 4$. Here we provide a different example for a 3×3 switch where a non-anticipative tracking policy does not exist.

Example : Consider a 3×3 switch. Suppose that the serving discipline of every link under the fluid policy is determined by the number of packets in each buffer and the priority of the packets as follows.

- 1. If less that one packets are queued in each link, these packets are served with equal rates. If there are links for which more than one packets are queued, all these links - and only these links - are served with equal rate.

2. In each of the links, higher priority packets are served first.

Suppose that packets $p_1, \ldots, p_9$ arrived at the beginning of the first time slot. Buffered packets of the fluid and packetized policy are depicted in Fig. 5. The figure shows the buffering at the beginning of first five time slots. In every input there are three parallel buffers (virtual queues) corresponding to the three outputs. As it is illustrated there are 9 packets, one in each virtual queue at time 1. Each of these packets have distinct input/output pairs and all have the same priority. Let matrix $Q(k)$ specify the buffered packets under the packetized policy at time k. That is, element $q_{ij}(k)$ of $Q(k)$ specifies the set of packets that are buffered at input port i and are destined for output port j under the packetized policy. Assume that

$$Q(1) = \begin{bmatrix} p_1 & p_2 & p_3 \\ p_4 & p_5 & p_6 \\ p_7 & p_8 & p_9 \end{bmatrix}.$$

According to rule 1 above, the service matrices of the fluid policy will be,

$$w(1) = w(2) = \begin{bmatrix} 1/3 & 1/3 & 1/3 \\ 1/3 & 1/3 & 1/3 \\ 1/3 & 1/3 & 1/3 \end{bmatrix}.$$

Without loss of generality, assume that the tracking policy selects the following two permutation matrices for the first two time slots.

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

Now assume that at the beginning of time slot 3, six new packets $p_{10} \dots p_{15}$ arrived, so that the backlogged packets under the packetized policy at the beginning of slot 3 are,

$$Q(3) = \begin{bmatrix} p_{10} & p_{11} & p_3 \\ p_4 & p_{12} & p_{13} \\ p_{14} & p_8 & p_{15} \end{bmatrix}.$$

Hence in the figure at time slot 3, we have one packet in every queue under the packetized policy. The situation is different for the fluid policy, 1/3 of the previously arrived packets remain, and there are six new arrivals as well. Assume that $p_{10}, ..., p_{15}$ have equal priority, higher than the priority of the previous packets and hence, according to rule 2, they are placed ahead of the previous arrivals in the fluid policy queues. Based on rule 1, the serving rate of the fluid policy becomes,

$$w(3) = \begin{bmatrix} 1/2 & 1/2 & 0 \\ 0 & 1/2 & 1/2 \\ 1/2 & 0 & 1/2 \end{bmatrix}.$$

In order to ensure proper tracking, the tracking policy should also serve a feasible set of these high priority packets. For instance let,

$$J(3) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

At the beginning of time slot 4, assume that six new packets arrive, all with low priority, so that the queue matrix becomes,

$$Q(4) = \begin{bmatrix} 0 & p_{11}, p_{16} & p_3, p_{17} \\ p_4, p_{18} & 0 & p_{13}, p_{19} \\ p_{14}, p_{20} & p_8, p_{21} & 0 \end{bmatrix}.$$

Based on rule 1, the fluid policy would select the following serving rate matrix,

$$w(4) = \begin{bmatrix} 0 & 1/2 & 1/2 \\ 1/2 & 0 & 1/2 \\ 1/2 & 1/2 & 0 \end{bmatrix}.$$

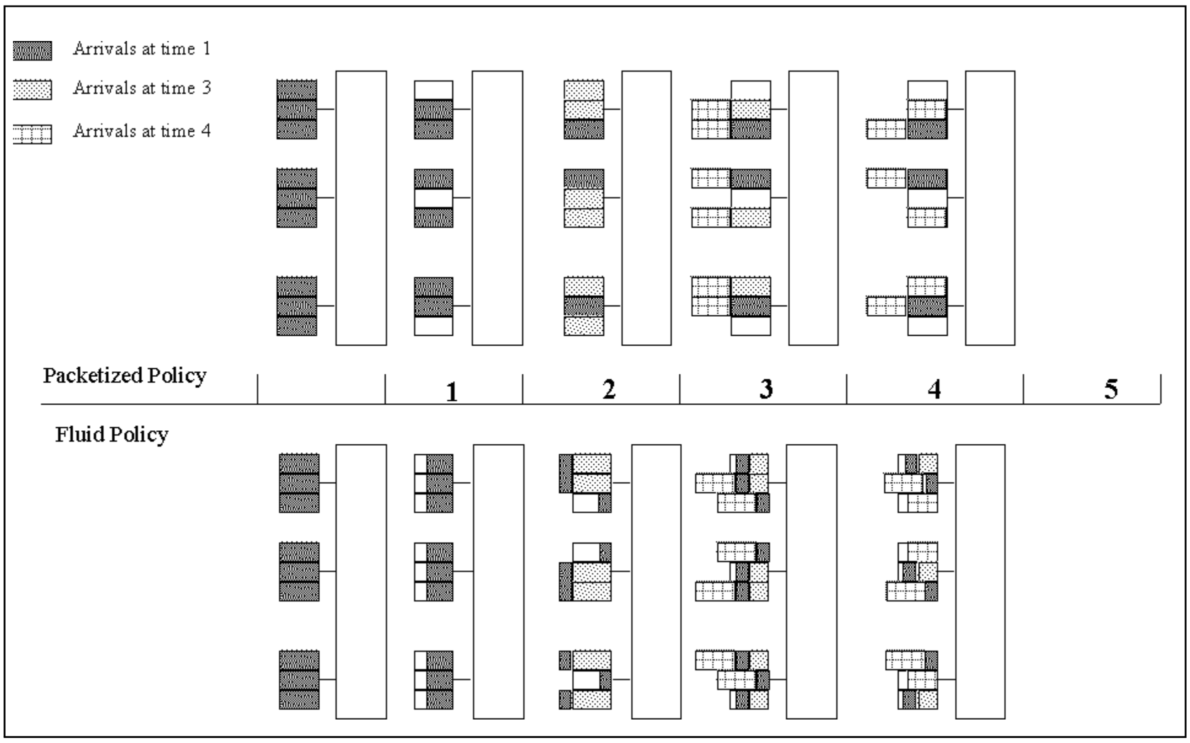

Fig. 5. The backlogged packets for the fluid and packetized tracking policy.

Therefore, at the beginning of time slot 5 six packets, with indices (3,4,8,11,13,14) are fully served under the fluid policy. However, by that time the packetized policy can at most, serve three of them. We assume that the tracking policy serves packets p_{11}, p_{13}, p_{14}, and there remain packets p_3, p_4, p_8 unserved. Similarly, it can be shown that for any of the other choices of $J(3)$, there is another possible set of new arrivals that makes it impossible for the packetized policy to track the fluid policy.

Since as the previous example shows it is impossible to construct non-anticipative policies in the general case, we are motivated to seek for heuristic algorithms with good but not perfect tracking properties. The design of the heuristic relies on two main concepts, port based tracking and critical links, that are discussed below. We design a simple tracking policy based on these concepts and illustrate its performance using simulation results.

A. Port Based Tracking

One way to implement a packetized tracking policy can be based on finding optimal weighted matchings in bipartite graphs. At slot k, with each input-output port pair (i, j), a weight, $t_{ij}(k)$, is associated. This weight represents the amount of work on input-output port pair (i, j), by which the fluid policy is ahead of the tracking policy, up to slot k. That is,

$$t_{ij}(k) = \max\left(\sum_{l=1}^{k} \left(w_{ij}(l) - J_{ij}(l)\right), 0\right).$$

We call these weights the tracking weights, $t_{ij}(k)$.

We view the switch as a bipartite graph, with nodes the input and output ports of the switch. The weight of link (input-output port pair) (i, j) is $t_{ij}(k)$. The weight of node (port) i, $v_i(k)$, is the sum of the weights of the links that emanate from or terminate at that node. That is, for an input port i and an output port j, we have respectively.

$$v_i(k) = \sum_{j=1}^{N} t_{ij}(k), \tag{13}$$

$$v_j(k) = \sum_{i=1}^{N} t_{ij}(k). \tag{14}$$

Each sub-permutation matrix $J(k)$ defining the tracking policy at slot k, corresponds to a matching in the bipartite graph. The selection of the appropriate matching can be based on the link and node weights introduced above. Ideally, the matchings should be chosen so that all these weights remain equal to zero.

Bipartite matching algorithms have been extensively used in switch scheduling and they are either based on the maximum link (edge) weighted matching or on maximum matching (that is, a matching that maximizes the number of links included in the matching) algorithms. Maximum link weighted matching algorithms are complex and computationally intensive, while maximum matching algorithms often have poor performance. We concentrate here on algorithms based on optimal vertex weight related matchings [14]. As will be seen, algorithms based on these matchings have good performance. Also, their computational cost is not very high. Two possible candidate matchings are those that satisfy the following optimization criteria.

Maximum node weight sum : With this criterion, the matching whose node weight sum is maximum is chosen. Hence, at every step it is attempted to find the matching whose node weight sum "lags" the most from the desired schedule. Since packets will be transmitted on the links of this matching, its "cost" will be reduced in the next time slot.

Maximum Lexicographic Order of Node Weights: In this approach, with each matching we assign a vector of node weights. Nodes that are included in the matching are assigned their weights $\nu_i(k)$, while nodes not included in the matching are assigned weight 0. We then select the matching whose assigned vector is maximal in the lexicographic order (max-min fair) [2, Section 6.5.2]. In this approach, it is attempted to select the matching whose nodes have individually large weights and hence are lagging the worst from the desired schedule.

It can be shown that both the above mentioned criteria are equivalent. This is due to the special structure of bipartite graphs. In the next lemma we prove the equivalence of the two criteria.

Lemma 3: Any matching that maximizes the node weight sum, maximizes the lexicographic order of weights and vice versa.

Proof: let $\overline{M}_1$ be a matching that maximizes the node weight sum and $\overline{M}_2$ a matching that maximizes the lexicographic order of weights. Let also M_1, M_2 be their respective set of nodes. We will show that for any node $i \in M_1 - M_2$, there is a node $j \in M_2 - M_1$, such that $\nu_i(k) = \nu_j(k)$ and vice versa. This implies that the two matchings have the same node weight sum, and that they are equal in the lexicographic order.

Consider the graph G that consists of links that belong to one and only one of the two matchings. Note that the maximum degree of a vertex in G is two. Let $i \in M_1 - M_2$. Then, degree of node i in G is one. Therefore, there is an (undirected) path in G that starts from node i and ends to a node j with degree one. We concentrate on this path.

The number of nodes in this path is even or odd. If it is even then it follows from the definition of G that the last node in the path belongs to M_1, while all intermediate nodes belong to both matching. Thus, if we replace the alternative set of links in the path belonging to $\overline{M}_2$ with those belonging to $\overline{M}_1$, two more vertices will be included to $\overline{M}_2$, and this contradicts with the optimality assumption of $\overline{M}_2$. Therefore, this case is impossible.

If the number of the nodes in the path is odd, then $j \in M_2 - M_1$. In this case, we necessarily have $v_j(k) = v_i(k)$. Indeed, if $v_j(k) < v_i(k)$ then as in the previous paragraph we can construct a matching that is better in the lexicographic order than $\overline{M}_2$. If on the other hand $v_j(k) > v_i(k)$, then we can similarly construct a matrix that is better than $\overline{M}_1$ in the node weight sum criterion. ■

We call an optimal matching based on the above criterion, Maximum Node Matching (MNM). Notice that MNM is different form conventional maximum weighted matching, since the latter maximizes the sum of the

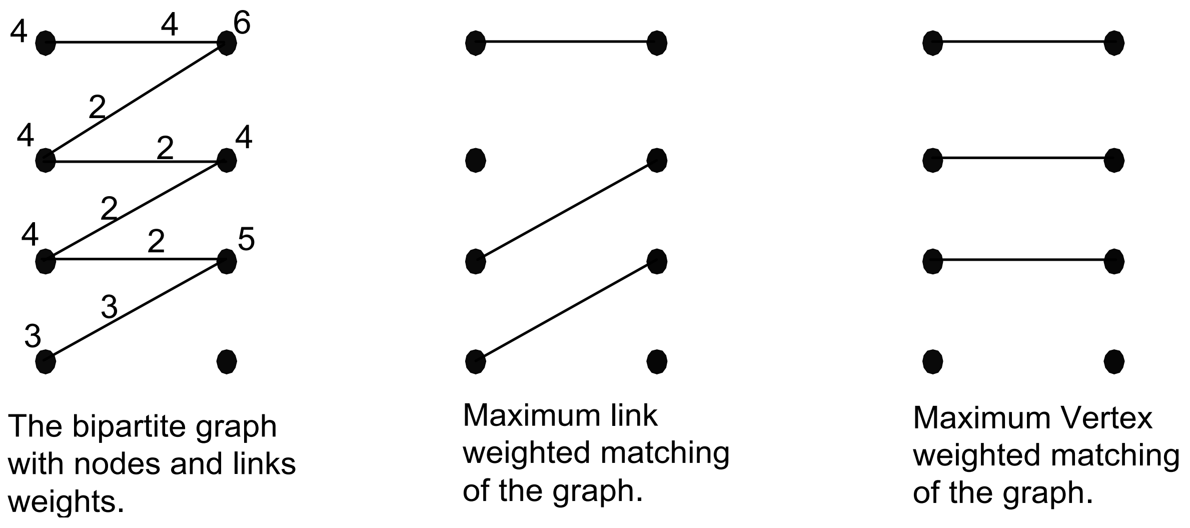
4
4
6
2
4
2
4
2
4
2
5
3
3
The bipartite graph with nodes and links weights.
Maximum link weighted matching of the graph.
Maximum Vertex weighted matching of the graph.

Fig. 6. Example of node and link weight matching in bipartite graphs.

weights of the links involved in the matching, while MNM maximizes the sum of the weights of the nodes involved in the matching. The difference is illustrated in Fig.6. Next, we need to have an algorithm for finding a Maximum Node Matching of a bipartite graph. Note first that MNM should be a maximum matching. To see this, assume that the algorithm employed to find the maximum matching is the augmented path algorithm for the associated maximum flow problem in an extended network [4]. This algorithm starts with an initial matching (flow on the extended network) and then at each iteration finds a flow augmenting path and a new flow in the extended network. Observe that at each iteration, due to the fact that the graph is bipartite, all the nodes in the original matching are still nodes of the new matching. Hence, assuming that the algorithm uses the MNM matching as its initial matching, the final matching includes all MNM nodes. Moreover, the final matching cannot include additional nodes since then its node weight sum would be larger than that of the MNM matching.

Assume now that we find a maximum matching $\overline{M}_1$.We show next how to obtain a MNM, $\overline{M}^*$, from $\overline{M}_1$. Given any matching $\overline{M}$, define an "alternating path" to be a path in the bipartite matching such that of any two consecutive links in the path, one belongs to $\overline{M}$ and the other does not belong to $\overline{M}$. If $\overline{M}$ is a maximal matching, then it is easy to see that any alternating path such that one of its endpoints does not belong to $\overline{M}$, must contain an even number of links. Hence, the other endpoint of the path belongs to $\overline{M}$. The algorithm for finding an MNM is based on the following lemma

Lemma 4: A maximum matching $\overline{M}_1$ is an MNM, if and only for any alternating path with endpoints i belonging to $\overline{M}_1$ and j not belonging to $\overline{M}_1$,we have

$$v_i(k) \geq v_j(k).$$

Proof: Let $\overline{M}_1$ be an MNM. If there is an alternating path such that $v_i(k) < v_j(k)$, then we can replace the links of $\overline{M}_1$ in the path with the links in the path not belonging to $\overline{M}_1$. The resulting matching will have larger node weight sum than $\overline{M}_1$, which contradicts the assumption that $\overline{M}_1$ is MNM.

Assume now that for any alternating path with endpoint $i \in \overline{M}_1$ we have $v_i(k) \geq v_j(k)$. Let $\overline{M}^*$ be an MNM matching. Let G be the graph whose links are the links that belong to one and only one of $\overline{M}_1$ and $\overline{M}^*$. Since both $\overline{M}^*$ and $\overline{M}_1$ are maximum matchings, using arguments similar to those used in the proof of Lemma 3, it can be seen that in G there can be either alternating cycles or alternating paths with even number of links. If G contains only cycles, then $\overline{M}_1$ has the same nodes as $\overline{M}^*$ and hence the same node weight sum. Assume now that there is a path in G such that one of its end nodes, i, belongs to $\overline{M}_1$ and the other, j, in $\overline{M}^*$. Since $\overline{M}^*$ is MNM, we must have $v_i(k) \leq v_j(k)$. Since for any alternating path we have $v_i(k) \geq v_j(k)$, we conclude that $v_i(k) = v_j(k)$. We conclude that the node weights of $\overline{M}_1$ are equal to the node weights of $\overline{M}^*$ in the lexicographic order and hence they again have the same node weight sum. ■

Based on Lemma 4 we have the following algorithm for finding an MNM matching.

Matching Algorithm:
1. Apply any maximum matching algorithm to obtain an initial maximum matching $\overline{M}_1$.
2. Sort all vertices not in M_1 by their weights, and mark all of them as unexplored.
3. If among the alternating paths originating from the unexplored node, i, with highest weight, there is one such that its other endpoint, j, (belonging necessarily to $\overline{M}_1$) has smaller weight than the weight of node i, then
 (a) replace in $\overline{M}_1$, the links in the path belonging to $\overline{M}_1$ with the links in the path not belonging to $\overline{M}_1$.
 (b) Remove node i from the unexplored set and include node j in the unexplored set. Else,
 (c) Remove node i from the unexplored set.
4. If the set of unexplored nodes is empty, then $\overline{M}_1$ is an MNM. Else go to step 3.

So far we have assumed that all lagging links (those with positive weights) are included in the bipartite graph, and that the weight of every node is the sum of weights of the links that emanate from the node. In the next section, we will discuss a trimming mechanism that enables us to modify the weight of nodes and to exclude some of the graph links so that the matching algorithm can come up with better assignment configurations.

B. Critical Ports and Links

A critical port is a port for which a packet 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 kth time slot. Assume that there are two packets, one that needs to be transmitted from node i to node j_1 and the other from node i to node j_2. Assume also that both have deadline $k+2$. Note that if we do not schedule any of these packets, no deadline will be missed in the kth 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, a 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 $k+p$. Note that we are stating a sufficient condition. In other words, there might be some critical ports that cannot be detected well in advance using this criterion. However, for simplicity we concentrate on nodes that are critical according to the criterion defined above. Our goal is to detect critical nodes and increase their chance to be scheduled.

In case of the tracking policy, the deadlines of packets are implicitly given, and are equal to the end of the time slot that the packet departs the switch under the fluid policy. We may not know the deadlines in advance, since the future rate of every link under the fluid policy depends on the future arrivals, which in general are not known. Nevertheless, we may have an approximate deadline for every packet based on back-logged traffic or the average arrival rate of the links

The next issue is to set an appropriate inspection horizon. Suppose that we are at time k. To detect the critical links, we have to account for the packets that should be scheduled in the next p time slots. We call p the "inspection horizon". 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 port, otherwise we will miss the deadlines. To give priority to this node, we increase its weight by a constant, so that its weight exceeds all non-critical nodes weights. Therefore, these nodes are prioritized by the scheduler. To make sure that one of its critical links are scheduled, we remove all non-critical links that have a critical node as an endpoint. The algorithm for detecting critical nodes can be described as follows,

Critical Node Detecting Algorithm:
1. At every time step k, set $p = 1$.
2. For each node, i, find the number of packets that have to be sent in the next p time slots. These are the packets with deadlines at most $k + p$.
3. If for some node, i,the number of packets that should be sent in the next p time steps is larger than or equal to p, then i is critical. Moreover, the links emanating from i, over which at least a packet should be sent in the next p time slots are critical links
4. Increment p and go back to step 2, if $p \leq L_{\max}$ ($L_{\max}$ is the inspection horizon).

The computationally expensive part of this algorithm is step 2. At that step we need to estimate the work done by the fluid policy in the next p time slots. In general the estimate depends on future arrivals and cannot be computed exactly. An approximate value can be obtained by assuming that there are no new arrivals in the system. There are cases of course, as in the example described in Section III, where the design is done off-line and future arrival can be anticipated.

The scheduling process of a switch can be divided into two stages. In the first stage, the weight of the ports are calculated and the criticality of the ports is investigated. Once the service rates of the fluid policy are computed, the rest of computations for different ports of the switch can be done in parallel, 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 matching algorithm.

Finally we provide the scheduling algorithm that is based on the algorithms described above.

Scheduling Algorithm:
1. At every time slot k do the following steps.
2. Calculate node weights using (13), (14).
3. Insert all links with positive weights in the eligible links set.
4. Check for critical nodes and their associated critical links.
5. Increase the weight of every critical node to a value, C, where C is larger than the weights of all non-critical nodes. Remove all non-critical links of the critical nodes from the eligible links set.
6. Pass the weights of the nodes and the critical links to the matching algorithm. The result of the matching is the schedule for time slot k.

If there are several sessions that are transmitting packets in an input/output pair, the scheduling algorithm does not specify, which of them should be scheduled in the associated slot. The scheduler considers all these sessions as an aggregated session and works with the aggregate rate. Once a slot is assigned to an input/output pair, it is the responsibility of a local scheduler maintained at the input port to assign the space to one of the multiple sessions. In principle, any single link sharing algorithm may be used as local scheduler. Here we will use EDF scheduler. This hierarchal approach improves the scalability of the scheduler. Note that apart from the computations that may be needed to calculate rates under the fluid policy, the complexity of the scheduler does not depend on the number of sessions between every input/output pair, since all of them are considered as an aggregated session.

The proposed scheduling algorithm is used in next section to schedule fixed rate sessions, and its performance is evaluated through simulations.

C. Fix Rate Scheduler Simulation

We consider multiple fixed rate sessions arrive to all input ports of a switch. These sessions are similar to the periodic sessions introduced in Section III. Each session m, has an integer period p_m. In any interval $[kp_m,\ (k+1)p_m - 1]$ one slot should be assigned to session m. If the slot is not assigned, we assume that one packet of that session is discarded. In effect, we are assuming that real time sessions have strict deadlines, but can tolerate some packet loss. Moreover, the total capacity dedicated to real time sessions is less than the capacity of the switch. This does not necessarily mean that some of the capacity is wasted, since the remaining capacity could be dedicated to non-real time traffic. In fact, as will be seen, the capacity we considered allocated to real-time traffic in our simulations is much higher than the capacity normally assigned in today's networks.

The main input parameters to the simulation are the maximum port utilization and minimum overall utilization of the switch ports , u_M and u_m respectively, and the switch size N. We denote the first two parameters as the utilization pair, (u_M, u_m). In the first set of experiments, the inspection horizon $L_{\max}$ is considered as an input and its effect is studied, while in the rest of experiments it is set to a constant value. The sessions are generated as follows. A uniform random number generator is used to select the input and output ports for every session. The rate of a session is selected uniformly in the range of [1/1024, 68/1024], so that the period of sessions is from 15 to 1024 slots. If the selected rate is such that one of the port load exceeds the maximum port rate, then the rate is clipped so that the overall load of that port equals the maximum load. The period of the session is then set to the ceiling of the inverse of the resulting rate. The above process

TABLE II
HEURISTIC ALGORITHM PERFORMANCE FOR DIFFERENT INSPECTION HORIZONS(U=(0.85, 0.8), N=32).

$L_{\max}$	0% Ratio	10% Ratio
0	0.9322	0.9996
1	0.9732	0.9997
2	0.9786	0.9997
3	0.9797	0.9997
4	0.9803	0.9997
5	0.9800	0.9997

is repeated 10000 times. This does not mean that there are 10000 sessions in each session set, since in some of the attempts either the input or the output port are fully loaded. Once set of sessions is generated, if the resulting average utilization of the switch ports exceeds u_m it is accepted, otherwise it is discarded and another session set is formed. The minimum session rate is set to 1/1024, because for each session set, the simulation runs for 1024 time steps.

One of the main advantages of the heuristic algorithm is that its complexity is not a function of the number of sessions. The rate of all individual sessions with same input and output are added up and the arbiter looks at them as an aggregated session. Once a slot is assigned to a link, then there is a local scheduler that selects the session that is going to use that slot. In our case, we simply use an EDF scheduler for this purpose. If no slot is assigned to a session during one of its period interval, we assume that one of its packets is discarded. To study the effect of different parameters several experiments are carried out. Each experiment is specified by the values selected for switch size, utilization pair, and the inspection horizon. For each experiment 100 sets of sessions are generated, and for each session set 1024 time steps of simulation is performed. For every session, the percentage of discarded packets is calculated. The performance measure is the percentage of sessions with no discarded packet (%0 loss ratio), and the percentage of packets with %10 loss ratio. Three different aspects of the algorithm are studied, inspection horizon, switch size, and the utilization pair.

C.1 Inspection Horizon

We introduced the concept of critical links as a way to detect and increase the chance of the links and ports that are more urgent to be scheduled. Obviously this increases the complexity of the scheduling algorithm. In fact, the additional computation load is a function of the selected inspection horizon. In the first series of experiments, we study the effectiveness of this procedure and the appropriate values for inspection horizon. The switch size is set to 32 and the utilization pair to (0.85, 0.8). The results are given in table I. We can deduce that the detection of critical links can improve the capacity, and reduce the percentage of non-perfectly scheduled sessions by about 5%. Notice that for $L_{max} = 0$ (no check), 0.068 of sessions have discarded packets, while for $L_{max} = 1$, this reduces to 0.027.

C.2 Switch size

In this series of experiments, the inspection horizon is fixed to 5, and the utilization pair is (0.85, 0.8). Most of the heuristic algorithms provided for input queueing switches fail to give satisfactory result for moderate size switches [17]. The results of our simulation are given in table II. We also observe some degree of performance degradation as the switch size increases. However, in all cases the %0 ratio is around 0.98. In fact, if we decrease the network load, we can even get better results. This is a very important feature of the algorithm, since it is vital for the algorithm to perform well for larger switch sizes.

C.3 Utility

In this series of experiments the effect of utilization or switch load is investigated. The switch size is set to 32, which is a moderate size switch. The results are given in table III. As we expect the performance of the

TABLE III
Heuristic Algorithm Performance for different switch sizes(U=(0.85, 0.8)).

N	0% Ratio	10% Ratio
8	0.99	0.9999
16	0.98	0.9997
32	0.98	0.9997
64	0.98	0.9997

TABLE IV
Heuristic Algorithm Performance for different utility pairs (N=32).

U	0% Ratio	10% Ratio
(0.55, 0.5)	0.996	1
(0.65, 0.6)	0.993	0.9999
(0.75, 0.7)	0.989	0.9999
(0.85, 0.8)	0.980	0.9997
(0.95, 0.9)	0.952	0.9986

system degrades as a function of utilization. However, with the exception of the utilization pair (0.95, 0.9), which is very high for a realistic system, the percentage of sessions without any packet loss is above 98%.

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 useful in the design of high speed input queued switches, where they can aid in the design of policies that provide guaranteed service to different applications, and in TDMA-SS with multi-periodic sessions. The existence of a tracking policy is proved for the special case of 2×2 switches. For the general case of $N \times N$ switches a heuristic algorithm is provided.

The design of tracking policies for a general $N \times N$ switch is still an open question. The examples in Section V show that such a tracking policies cannot be designed without further constraints on the arrivals or on the policies themselves. This fact, together with the complexity that a perfect tracking policy might entail, justify the need for less complicated heuristic tracking policies, with good performance. The proposed heuristic algorithm is based on two useful notions, the Maximum Node Matching, and Critical Nodes. The scheduling is done in a hierarchical fashion. First the global scheduler selects the input-output pairs on which packets may be transmitted in a particular slot, and then a local scheduler assigns the slot to one of the sessions sharing the input-output pair. This approach makes the scheduler scalable in terms of the number of sessions. The simulation results are promising and illustrate that the algorithm can be useful in high speed networks and satellite switches where not only throughput but delay and jitter guarantees are desirable too.

References

[1] J.C.R. Benett and H. Zhang WF2Q: Worst-case Fair Weighted Fair Queueing *Proceedings of INFOCOM '96*, IEEE, March 1996.

[2] D. Bertsekas, R. Gallager, *Data Networks,* Prentice Hall, 1992.

[3] M.A. Bonuccelli, M.C. Clo. EDD Algorithm Performance Guarantee for Periodic Hard-Real-Time Scheduling in Distributed Systems. *IPPS/SPDP 1999*, San Juan, Peurtorico Rico, April 1999.

[4] T. H. Cormen, C. E. Leiserson, R. L. Rivest, "Introduction to Algorithms," Mc Graw Hill, 1990.

[5] S. Chuang, A. Goel, N. McKeown, B. Prabhakar *Proceedings of INFOCOM '99*, 1169–1178, IEEE, April 1999.

[6] R. Cruz. A calculus for network delay. I. Network elements in isolation. *IEEE Trans. on Information Theory*, 37(1):114–131, January 1991.

[7] R. Cruz. A calculus for network delay. II. Network Analysis. *IEEE Trans. on Information Theory*, 37(1):132–141, January 1991.

[8] A. Demers, S. Keshav, and S. Shenkar. Analysis and simulation of a fair queueing algorithm. *Proceedings of SIGCOM '89*, pages 1–12.

[9] L. Georgiadis, R. Guerin, and A. Parekh. Optimal multiplexing on single link: Delay and buffer requirements. *IEEE Transactions on Information Theory*, vol. 43, no. 5, p 1518–1535, Sep. 1997.

[10] 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.

[11] J. Giles, B. Hajek. Scheduling multirate periodic traffic in a packet switch Shaping. *Conf. on Info. Sci. and Systems at John Hopkins Univ*, 1997.

[12] S. Golestani. A self-clocked fair queueing scheme for broadband applications. *Proceedings of INFOCOM '94*

[13] T. Inukai An efficient SS/TDMA time slot assignment algorithm. *IEEE Transactions on Communications*, vol.27, pp. 1449-1455, 1979.

[14] A. Mekkittikul, N. McKeown. A Practical Scheduling Algorithm to Achieve 100% Throughput in Input-Queued Switches. *Proceedings of INFOCOM '98*, pp. 792-799.

[15] 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.

[16] 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.

[17] I.R. Philp. Scheduling real-time messages in packet-switched networks. *Ph.D. Dissertation Dept. Comp. Sci. Univ of Illinois at Urbana-Champaign* , 1997.

[18] H. Zhang, D. Ferrari. Rate-controlled service disciplines. *J. High Speed Networks*, vol.3, no.4,pp. 389-412, 1994.