

Low-Jitter Scheduling Algorithms for Deadline-Aware Packet Switches

Aditya Dua and Nicholas Bambos
Department of Electrical Engineering,
Stanford University
350 Serra Mall, Stanford, CA 94305
Phone: 650-725-5525, Fax: 650-723-4107
Email: {dua,bambos}@stanford.edu

Abstract—We study low jitter- scheduler design for deadline-aware input-queued (IQ) packet switches. We consider scheduling of traffic streams associated with service profiles, which reflect the inter-packet deadlines between packets constituting the stream. To make the NP-hard problem of scheduling with strict deadlines tractable, we use soft deadlines as a modeling tool, and study the scheduling problem with soft deadlines in a dynamic programming (DP) framework. We establish the optimality of a myopic scheduling policy for the canonical 2×2 crossbar switch. For bigger switches, we develop low-complexity approximations to the myopic policy (which is near-optimal), one based on the notion of neighborhood search, and two others based on convex relaxations of an integer programming problem. We demonstrate the efficacy of the proposed policies via simulations, employing goodput as a performance metric. A key feature of the proposed policies is that they do not require knowledge of traffic statistics (rate, periodicity etc.), rendering them robust and amenable to implementation.

Index Terms—Input-queued switches, Scheduling, Deadlines, Dynamic Programming, Myopic policy, Convex optimization.

I. INTRODUCTION

Real-time applications like multimedia streaming, video telephony etc. continue to gain popularity amongst internet users. These applications have stringent quality-of-service (QoS) requirements with regard to packet delays and jitter. Scheduling algorithms employed at packet switches play a key role in QoS provisioning for such applications.

Input queued (IQ) switches have received considerable attention, owing to their scalable architecture. Most research on IQ switch scheduling has focused on throughput and stability related issues. Several throughput optimal scheduling algorithms based on maximum weight matching (MWM) have been proposed ([1], [2] etc.) in the literature. This vast body of results, while important in its own right, does not address the question of QoS provisioning for deadline sensitive traffic.

Liu *et al.* [3] studied scheduling of multi-class periodic traffic flows through IQ switches. They conjectured that it is possible to schedule periodic traffic through an IQ switch so that each packet from a flow leaves the switch before the next packet of the flow arrives, provided the line utilization does not exceed unity. Giles *et al.* [4] (nested period scheduling), and Rai *et al.* [5] (uniform weighted round robin) proposed heuristic scheduling policies for periodic flows. Chang *et al.* proposed schemes for providing delay guarantees in IQ

switches based on the Birkhoff-von Neumann (BV) decomposition in [6], and based on the earliest deadline first (EDF) rule for load balanced switches in [7]. Keslassy *et al.* [8] studied low-jitter scheduling based on BV decomposition, and Li *et al.* [9] proposed a frame based scheduler with guaranteed delay and jitter bounds for leaky-bucket constrained traffic.

Our focus is on developing scheduling algorithms for traffic streams associated with *service profiles*. The service profile captures the inter-packet deadlines (IPD) between successive packets in a traffic stream; it determines the ideal inter-departure times of packets from the switch. A deviation from the service profile (jitter) results in missed packet deadlines, and hence QoS degradation at the receiver.

We do not make any assumptions on the periodicity or any other statistics of traffic streams. Traffic arriving to a switch can be bursty and aperiodic over different time-scales due to several reasons - bursty sources (for example, variable bit rate video), stream multiplexing, jitter induced by upstream switches etc. It is therefore crucial to design schedulers agnostic to periodicity assumptions.

The scheduling problem with strict deadlines is NP-hard. To surmount the associated computational complexity, we employ the notion of *soft deadlines* or flexible deadlines. Packets are allowed to violate these soft deadlines, while incurring a penalty for doing so. Packets also incur a penalty for being ahead of their deadlines. This prevents streams from receiving more service than they need to meet their delay requirements. Our goal is to design low-complexity schedulers which minimize aggregate soft deadline violation over all input streams, or equivalently, ensure low jitter for all output streams. Our modeling approach is akin to the *time/utility function* (TUF) approach introduced by Jensen *et al.* [10] to study scheduling in real-time operating systems.

The typical metric for delay in most prior work on scheduling is average queuing delay, which is measured on a *macro* time-scale. Our work represents a paradigm shift in conventional scheduler design because the focus is now on *micro* time-scales (on the order of packet deadlines). The goal is no longer to minimize the average delay per packet, but to ensure that *each* packet meets its deadline as closely as possible. The performance of scheduling algorithms in this setting is to be gauged by the fraction of packets which meet their deadlines,

or the *goodput*, rather than the throughput or average delay.

A. Organization of the paper

We construct a mathematical model for the scheduling problem with soft deadlines in Section II. We analyze this model for a 2×2 crossbar switch in a dynamic programming (DP) framework [11] in Section III. Our key result is that the optimal finite-horizon policy is *myopic*. We then develop three low-complexity approximations to the myopic policy for bigger switches (which is near-optimal) in Section IV. We show the efficacy of the proposed policies via simulations in Section V, and provide concluding remarks in Section VI.

II. MODEL CONSTRUCTION

Consider an IQ switch with virtual output queues (VOQs) at the input ports to prevent head-of-line blocking. Each VOQ is associated with a *traffic stream*, comprised of packets arriving to that VOQ. Traffic streams with deadline constrained packets can be described using a vector* $\mathbf{x} = (x^0, x^1, \dots)$, with $x^i \in \{0, 1\}$. We will call this the *expected service vector* (ESV) for a stream. Suppose that the k^{th} “1” in the traffic vector occurs at location m and the $(k+1)^{\text{st}}$ “1” occurs at location $m + d_k$. The interpretation is that if the stream is being serviced in accordance with its ESV, the inter-departure time between the k^{th} and $(k+1)^{\text{st}}$ packets is exactly d_k time-slots. Equivalently, d_k is the *inter-packet deadline* (IPD) between the k^{th} and $(k+1)^{\text{st}}$ packets. ESV formally embodies the notion of service profile discussed in Section I. We define the *cumulative expected service vector* (CESV) $\mathbf{X} = (X^0, X^1, \dots)$, with $X^n = \sum_{i=0}^n x^i$ being the number of packets of the stream which should ideally have departed the switch by the end of time-slot n .

Due to congestion caused by resource contention for multiplexed output links at the switch, a stream will not always receive service in accordance with its ESV. To quantify this effect, we associate with each traffic stream a *received service vector* (RSV) $\mathbf{y} = (y^0, y^1, \dots)$ with $y^i \in \{0, 1\}$. Then, $y^i = 1$ if the switch forwards a packet of this stream in the i^{th} time-slot, and 0 else. Similar to CESV, we define the *cumulative received service vector* (CRSV) $\mathbf{Y} = (Y^0, Y^1, \dots)$, with $Y^n = \sum_{i=0}^n y^i$. Y^n is the number of packets that have been forwarded from the stream until the end of time-slot n . Ideally, we would like $Y^n = X^n \forall n$. If $Y^n > X^n$, the stream has received excess service, is said to be *leading*. If $Y^n < X^n$, the stream is starved of service, and is said to be *lagging*. To capture the temporal evolution of a stream, we define its *deviation* as $D^n \triangleq Y^n - X^n$. The objective of the scheduler is to keep the deviation D^n as close to 0 as possible at all times, for every traffic stream. The idea is pictorially depicted in Fig. 1, on the left side.

Since we allow for deviations from expected service, we can think of packets as having *soft deadlines*. Packets which

miss their soft deadlines are not dropped, but are penalized for doing so. Packets are also penalized for being ahead of their deadlines. A “good” scheduler ought to minimize both the number and magnitude of soft deadline violations.

III. THE CANONICAL 2×2 SWITCH

In this section, we study the scheduling problem for the canonical 2×2 IQ switch. The input and output ports are labeled 0 and 1. VOQ Q_{ij} holds packets destined from input port i to output port j . In each time-slot, the switch can be set in configuration C_0 or C_1 . In configuration C_0 , input ports 0, 1 are connected to output ports 0, 1 respectively, while in configuration C_1 they are connected to output ports 1, 0 respectively. At most one packet can be transferred to each output port from one of the input ports in a time-slot (*speed-up* 1).

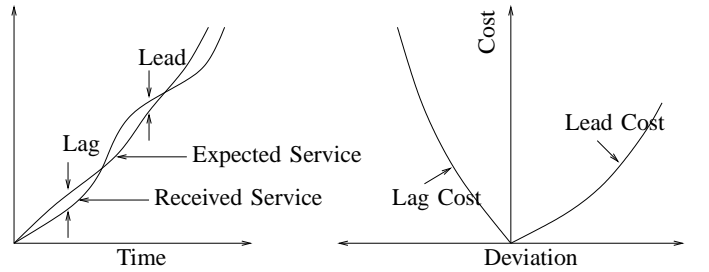


Fig. 1. The left side depicts the notion of expected and received service, and deviation from expected service. The right side depicts a typical deviation cost associated with a VOQ.

A. Cost structure

We define the **state** of system as the four-tuple $\mathbf{s} = (s_{00}, s_{11}, s_{01}, s_{10})$, where s_{ij} is the deviation associated with Q_{ij} . We denote $\mathbf{x}^n = (x_{00}^n, x_{11}^n, x_{01}^n, x_{10}^n)$, where x_{ij}^n is the n^{th} element of the ESV of the traffic stream associated with Q_{ij} . Similarly, $\mathbf{X}^n = (X_{00}^n, X_{11}^n, X_{01}^n, X_{10}^n)$ where X_{ij}^n is the n^{th} element of the CESV of the traffic stream associated with Q_{ij} . We denote $\mathbf{a} = (1, 1, 0, 0)$, $\mathbf{b} = (0, 0, 1, 1)$ and $\mathbf{\Delta} = \mathbf{a} - \mathbf{b} = (1, 1, -1, -1)$. With Q_{ij} , we associate the cost function $\phi_{ij}(k) = f_{ij}^+(k_+) + f_{ij}^-(k_-)$, which measures the cost associated with deviation $k \in \mathbb{Z}$, where $k_+ = \max(k, 0)$ and $k_- = -\min(k, 0)$. We assume that f_{ij}^+ (f_{ij}^-) is a convex, non-negative and non-decreasing (non-increasing) function. A typical cost function is depicted in Fig. 1, on the right side.

We define the *sum cost function* $\phi(\mathbf{s}) = \sum_{i=1}^2 \sum_{j=1}^2 \phi_{ij}(s_{ij})$, as the sum of deviation cost of all four VOQs.

B. Dynamic Programming (DP) formulation

We will consider a finite time-horizon of N time-slots. To facilitate exposition, we will assume that all VOQs are infinitely backlogged, making them potential scheduling candidates over the entire time-horizon N . We define an admissible policy $\Pi_N \in \{C_0, C_1\}^N$ as a sequence of switch configurations in time-slots $0, \dots, N-1$. Letting $\mathbf{s}_{\Pi_N}^n$ denote the state

*Throughout this paper, vectors are denoted in **boldface**.

at the beginning of time-slot n under policy Π_N , the objective is to find policy Π_N^* satisfying

$$\Pi_N^* = \arg \min_{\Pi_N} \left\{ \Phi^N(\Pi_N) = \sum_{n=1}^N \phi(\mathbf{s}_{\Pi_N}^n) \right\}, \quad (1)$$

given the initial state \mathbf{s}^0 . We will adopt the methodology of *dynamic programming* to compute the optimal policy Π_N^* .

Let us first examine the evolution of the state under an admissible policy Π_N . In state \mathbf{s}^n at the beginning of time-slot n , Π_N can either choose configuration C_0 or C_1 . If Π_N chooses C_0 , the deviations of Q_{00} and Q_{11} either stay the same or increase by 1, depending on whether x_{00}^n and x_{11}^n are 0 or 1, respectively. The deviations of Q_{01} and Q_{10} either stay the same or decrease by 1, depending on whether x_{01}^n and x_{10}^n are 0 or 1, respectively. Thus, the state of the system changes from $\mathbf{s}^n = (s_{00}^n, s_{11}^n, s_{01}^n, s_{10}^n)$ to $(s_{00}^{n+1} + 1 - x_{00}^n, s_{11}^{n+1} + 1 - x_{11}^n, s_{01}^n - x_{01}^n, s_{10}^n - x_{10}^n)$, or in vector notation $\mathbf{s}^{n+1} = \mathbf{s}^n + \mathbf{a} - \mathbf{x}^n$. By the same token, if Π_N chooses C_1 in time-slot n , the new state is $\mathbf{s}^{n+1} = \mathbf{s}^n + \mathbf{b} - \mathbf{x}^n$.

Let $V^n(\mathbf{s})$ denote the *cost-to-go* at the beginning of time-slot n , starting in state \mathbf{s} . $V^n(\mathbf{s})$ is the cost incurred by Π_N^* from time-slot n to time-slot N , starting in state \mathbf{s} in time-slot n . By the *principle of optimality*, $V^n(\mathbf{s})$ can be computed from the following DP equations for $n = 1, \dots, N$:

$$V^n(\mathbf{s}) = \min \left\{ \underbrace{\Omega^{n+1}(\mathbf{s} + \mathbf{a} - \mathbf{x}^n)}_{\text{Configuration } C_0}, \underbrace{\Omega^{n+1}(\mathbf{s} + \mathbf{b} - \mathbf{x}^n)}_{\text{Configuration } C_1} \right\}, \quad (2)$$

and the boundary conditions $V^N(\mathbf{s}) = 0 \forall \mathbf{s}$, where

$$\Omega^n(\mathbf{s}) \triangleq V^n(\mathbf{s}) + \phi(\mathbf{s}). \quad (3)$$

Now define the *decision function*

$$\gamma^n(\mathbf{s}) \triangleq \Omega^{n+1}(\mathbf{s} + \mathbf{a} - \mathbf{x}^n) - \Omega^{n+1}(\mathbf{s} + \mathbf{b} - \mathbf{x}^n). \quad (4)$$

Configuration C_0 is optimal in state \mathbf{s} in time-slot n if $\gamma^n(\mathbf{s}) \leq 0$, and configuration C_1 is optimal else.

C. Properties of the optimal policy

We now explore some structural properties of the optimal scheduling policy Π_N^* . The two possible states in time-slot $n + 1$ starting in state \mathbf{s} in time-slot n are $\mathbf{s} + \mathbf{a} - \mathbf{x}^n$ and $\mathbf{s} + \mathbf{b} - \mathbf{x}^n$, which differ by $\mathbf{a} - \mathbf{b} = \mathbf{\Delta}$ (for any policy). The following results[†] become important in the light of this observation.

Lemma 1: $\phi(\mathbf{s} + \mathbf{\Delta}) - \phi(\mathbf{s}) \geq \phi(\mathbf{s}) - \phi(\mathbf{s} - \mathbf{\Delta}), \quad \forall \mathbf{s}$.

Lemma 2: $V^n(\mathbf{s} + \mathbf{\Delta}) - V^n(\mathbf{s}) \geq V^n(\mathbf{s}) - V^n(\mathbf{s} - \mathbf{\Delta}), \quad 0 \leq n \leq N, \quad \forall \mathbf{s}$.

Corollary 1: $\Omega^n(\mathbf{s} + \mathbf{\Delta}) - \Omega^n(\mathbf{s}) \geq \Omega^n(\mathbf{s}) - \Omega^n(\mathbf{s} - \mathbf{\Delta}), \quad 0 \leq n \leq N, \quad \forall \mathbf{s}$.

Corollary 2: $\gamma^n(\mathbf{s} + \mathbf{\Delta}) \geq \gamma^n(\mathbf{s}), \quad 0 \leq n \leq N, \quad \forall \mathbf{s}$.

Given the initial state \mathbf{s}^0 , we say that a state \mathbf{s} is *reachable* in time-slot n if there exists a sequence of configurations in time-slots $0, \dots, n - 1$ which drive the switch to state \mathbf{s} in

time-slot n . The reachable states in time-slot n constitute the set $\mathcal{S}_n = \{\mathbf{s}_k^n = \mathbf{s}^0 + k\mathbf{a} + (n - k)\mathbf{b} - \mathbf{X}^n, k = 0, \dots, n\}$. The state \mathbf{s}_k^n is reached if the optimal policy Π_N^* chooses C_0 in k of the time-slots from $0, \dots, n - 1$, and C_1 in the remaining $n - k$ time-slots. The states reachable in time-slot $n + 1$ starting from state \mathbf{s}_k^n in time-slot n are $\mathbf{s}_k^n + \mathbf{a} - \mathbf{x}^{n+1} = \mathbf{s}_{k+1}^{n+1}$ (if C_0 is chosen), and $\mathbf{s}_k^n + \mathbf{b} - \mathbf{x}^{n+1} = \mathbf{s}_k^{n+1}$ (if C_1 is chosen). Equivalently, we can identify the state in time-slot n by the index k , which increases by 1 in the next time-slot if C_0 is chosen and remains the same if C_1 is chosen. We now have the following key property:

Lemma 3 (Threshold Property): For $n = 0, \dots, N$, there exists a $k_n^* \leq n + 1$ such that $C^n(\mathbf{s}_k^n) = C_0 \forall k \leq k_n^*$ and $C^n(\mathbf{s}_k^n) = C_1 \forall k > k_n^*$.

The next result states that for fixed n , Ω^n achieves its minimum at k_n^* .

Lemma 4 (Minima of Ω^n): $\arg \min_{0 \leq k \leq n} \Omega^n(\mathbf{s}_k^n) = k_n^*$.

Equipped with the properties of Ω^n and γ^n enumerated above, we are ready to state the most important result regarding the optimal policy Π_N^* , which is the optimality of a *myopic* policy. What does it mean for the optimal policy to be myopic? It means that the optimal decision in state \mathbf{s} in time-slot n can be made greedily by merely choosing the configuration with the lowest ‘‘instantaneous cost’’, rather than solving the DP equations over the horizon n, \dots, N to compute the optimal decision. Equivalently, the result of one joint N -period optimization is the same as the result of N successive one-period optimization problems.

Theorem 1 (Optimality of Myopic Policy): The finite horizon optimal policy Π_N^* is a myopic policy.

Is the optimality of a myopic policy a useful property? Yes. Computing the optimal policy over an N -period horizon from the DP equations in (2) requires $\mathcal{O}(N^2)$ operations, while computing the myopic policy requires only $\mathcal{O}(N)$ operations. Also, as we shall see in the case of bigger switches, it is easier to design heuristics to approximate myopic policies.

D. Beyond 2×2 : Modular switches

Our results for a 2×2 switch are important in the context of modular switches, which can be realized by interconnecting 2×2 switches in a cascade fashion [13] (for example, Clos networks). The optimal policy Π_N^* can be used for each 2×2 element. Low jitter at each stage of the switch (ensured by Π_N^*) translate into low overall jitter for the switch.

IV. HEURISTIC SCHEDULING POLICIES

We have established that the optimal finite-horizon scheduling policy for a 2×2 switch is a myopic policy. Extensive numerical experiments showed that a myopic policy, although not optimal, is near-optimal for $K > 2$. However, evaluating the myopic policy is intractable even for moderately large K , owing to the explosion in the size of the decision space (it grows as $K!$). In a special case (quadratic cost functions), evaluating the myopic policy reduces to the problem of finding the maximum weight matching (MWM) on a bipartite graph,

[†]Due to space constraints, we omit all proofs and refer the reader to [12].

or equivalently, finding the configuration vector with the largest projection on the (negative of) the system state vector (comprised of deviations of all VOQs) [14]. However, MWM, with a computational complexity of $\mathcal{O}(K^3)$, is also quite unfriendly from an implementation perspective. Motivated by our discussion, we will now focus on developing low-complexity approximations to the myopic policy.

A. Neighborhood search

The neighborhood search heuristic is based on the assumption that the optimal configuration of the switch cannot change “drastically” from one time-slot to another [2]. We can think of a switch configuration as a permutation $\pi : \{0, \dots, K-1\} \rightarrow \{0, \dots, K-1\}$, such that input port i is connected to output port $\pi(i)$ in configuration π . A “neighbor” of permutation π is constructed by connecting input port i to output port $\pi(j)$ and input port j to output port $\pi(i)$, for some $i \neq j$. We denote the set of neighbors of configuration π by \mathcal{N}_π . Given the current system state and switch configuration π , the neighborhood search algorithm searches for the lowest-cost configuration over the set of configurations $\{\pi\} \cup \{\mathcal{N}_\pi\}$. The per-time-slot complexity is a tractable $\mathcal{O}(K^2)$.

B. Convex relaxation A

Computing the myopic policy is equivalent to solving an integer-programming (IP) problem in each time-slot, which can be formulated in time-slot n as follows:

$$\begin{aligned} & \text{minimize} \quad \sum_{i=0}^{K-1} \sum_{j=0}^{K-1} \phi_{ij}(s_{ij}^n - x_{ij}^n + \alpha_{ij}) \quad \text{subject to} \\ & \sum_{i=0}^{K-1} \alpha_{ij} = 1, \quad \sum_{j=0}^{K-1} \alpha_{ij} = 1, \quad \alpha_{ij} \in \{0, 1\}, \quad \forall i, j. \end{aligned}$$

α_{ij} is 1 if input port i is connected to output port j , and 0 else. While the above IP is NP-hard, we can “relax” the constraint $\alpha_{ij} \in \{0, 1\}$ to $\tilde{\alpha}_{ij} \in [0, 1]$ to obtain a constrained convex optimization problem. A convex formulation is attractive since computationally efficient methods are available for obtaining the solution [15]. The relaxation will have non-integer solutions in general. How do we convert this fractional solution into a valid switch configuration? For each input port i , we can think of the solution $(\tilde{\alpha}_{i,0}^*, \dots, \tilde{\alpha}_{i,N-1}^*)$ as a “preference list”. We map input port i to output port j if $j = \arg \max_k \tilde{\alpha}_{ik}^*$. Contentions can be resolved randomly or by port numbers.

C. Convex relaxation B for quadratic costs

While convex relaxation A was applicable to arbitrary choice of convex cost functions, we now restrict our attention to quadratic cost functions, that is, $\phi_{ij}(k) = k^2$. In particular, we solve the following convex optimization problem:

$$\begin{aligned} & \text{minimize} \quad \sum_{i=0}^{K-1} \sum_{j=0}^{K-1} (s_{ij}^n - x_{ij}^n + \beta_{ij})^2 \quad \text{subject to} \\ & \sum_{i=0}^{K-1} \beta_{ij} = 1, \quad \sum_{j=0}^{K-1} \beta_{ij} = 1, \quad \forall i, j. \end{aligned}$$

Note that we do not impose a constraint of the form $\beta_{ij} \in [0, 1]$. It can be shown (using Lagrange multipliers) that the solution to the above problem is given by

$$\begin{aligned} \beta_{ij}^* &= -\underbrace{(s_{ij}^n - x_{ij}^n)}_{\tilde{s}_{ij}^n} + \frac{1}{K} (\mathcal{A}_i + \mathcal{B}_j - \mathcal{S} + 1) \\ \mathcal{A}_i &= \sum_{j=0}^{K-1} \tilde{s}_{ij}^n, \quad \mathcal{B}_j = \sum_{i=0}^{K-1} \tilde{s}_{ij}^n, \quad \mathcal{S} = \frac{1}{K} \sum_{i=1}^{K-1} \sum_{j=0}^{K-1} \tilde{s}_{ij}^n. \end{aligned}$$

We can use approach similar to the one used for convex relaxation A to convert our fractional solution into a valid switch configuration.

V. SIMULATION RESULTS

In this section, we study the performance of the proposed policies via simulations. All our results are for a 4×4 IQ switch. We assumed that the IPDs of the input stream at the i^{th} VOQ at the j^{th} input port are geometrically distributed with parameter p_{ij} . We considered two different loading scenarios — (i) Uniform, that is, $p_{ij} = p \forall i, j$, and (ii) Diagonal heavy, that is, $p_{ii} = p$, $p_{ij} = p'$, $j \neq i$, such that $p > p'$. The total load per input port is $\ell = 4p$ for (i), and $\ell = p + 3p'$ for (ii).

We evaluated the performance of the myopic policy, and the three proposed approximations to the myopic policy, viz., neighborhood search, and convex relaxations A and B. We benchmarked performance against the round-robin (RR) scheduler, which chooses from amongst the 4! possible configurations in cyclic fashion, and the randomized (RAND) scheduler, which chooses one of the 4! configurations uniformly at random. We chose quadratic deviation cost functions, that is $\phi_{ij}(k) = k^2$, for all VOQs. Recall that in this case, computing the myopic policy reduces to the problem of finding the configuration vector with the largest projection on the (negative of) the current state vector. We call the myopic policy the **MaxProj** policy in this case. For diagonal heavy loading, we also considered a *biased* maximum projection (**bMaxProj**) policy, which uses “steeper” cost functions for the more heavily loaded diagonal VOQs, when they are lagging. In particular, we used $\phi_{ii}(k) = \lambda k^2$, $k < 0$ for the diagonal VOQs, where $\lambda > 1$ is the *bias* of bMaxProj.

For uniform loading, we varied p to vary ℓ from 0.9 to 0.98. For diagonal heavy loading, we fixed $p_{ij} = 0.2$, $i \neq j$, for a total off-diagonal load of 0.6. We varied p_{ii} from 0.3 to 0.38, to vary ℓ from 0.9 to 0.98.

We used *goodput* (fraction of packets which successfully meet their deadline constraints) and *average delay per packet* or *lateness* (computed for packets which miss their deadline) as metrics to contrast the performance of different schedulers. Fig. 2 and Fig. 3 respectively depict the goodput and delay performance of different scheduling algorithms under uniform loading. Fig. 4 and Fig. 5 depict the corresponding performance graphs under diagonal heavy loading. In both cases, MaxProj, and all the proposed heuristics comfortably outperform RR and RAND with respect to both metrics. Observe that

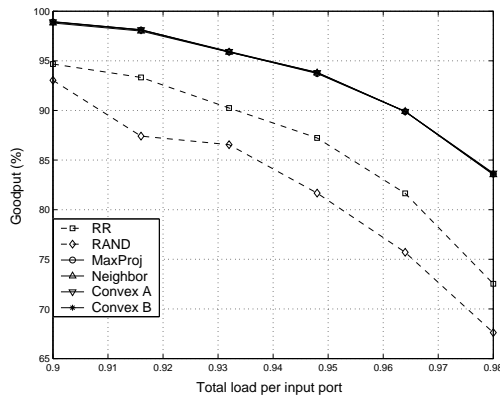


Fig. 2. Goodput v/s Load per input port (uniform loading)

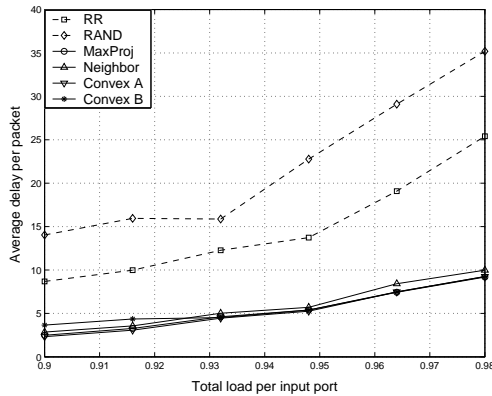


Fig. 3. Average delay v/s Load per input port (uniform loading)

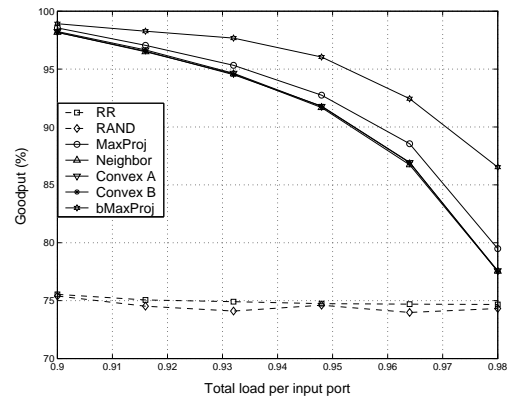


Fig. 4. Goodput v/s Load per input port (diagonal heavy loading)

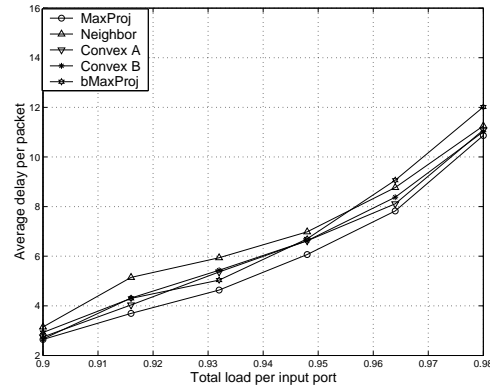


Fig. 5. Average delay v/s Load per input port (diagonal heavy loading)

the performance of MaxProj and the proposed heuristics is almost indistinguishable under uniform loading. However, under non-uniform loading, MaxProj performs marginally better than its approximations. The delay performance of neighborhood search is always slightly worse than the heuristics based on convex relaxations. Finally, using bMaxProj provides a significant improvement in goodput, albeit at the expense of a nominal increase in average delay.

VI. CONCLUSIONS

In this paper, we studied low-jitter scheduler design for deadline-aware crossbar packet switches, within a DP framework. We considered scheduling of traffic streams associated with service profiles, which capture the inter-packet deadline constraints between packets in the stream. Our work represents a departure from conventional scheduler design principles, with an emphasis on satisfying per-packet deadline constraints, rather than long-run throughput or average delay requirements.

REFERENCES

- [1] N. McKeown, A. Mekkittikul, V. Anantharam and J. Walrand, "Achieving 100% throughput in an input-queued switch", *IEEE Trans. Communications*, vol. 47, no. 8, pp. 1260-1267, Aug. 1999.
- [2] P. Giaccone, B. Prabhakar and D. Shah, "Randomized scheduling algorithms for high-aggregate bandwidth switches", *IEEE Journal on Sel. Areas in Commun.*, vol. 21, no. 4, pp. 546-559, May 2003.

- [3] I.R. Philip and J.W.S Liu, "SS/TDMA scheduling of real-time periodic messages", *Intl. Conf. on Telecomm. Systems*, pp. 244-251, Mar. 1996.
- [4] J. Giles and B. Hajek, "Scheduling multirate periodic traffic in a packet switch", *CISS 1997*, Baltimore, MD, Mar. 1997.
- [5] I.A. Rai and M. Alanyali, "Uniform weighted round robin scheduling algorithms for input queued switches", *IEEE ICC 2001*, pp. 2028-2032, Helsinki, Finland, Jun. 2001.
- [6] C.S. Chang, W.J. Chen and H.Y. Huang, "Birkhoff-von Neumann input-buffered crossbar switches for guaranteed-rate services", *IEEE Trans. Commun.*, vol. 49, no. 7, pp. 1145-1147, Jul. 2001.
- [7] C.S. Chang, D.S. Lee and Y.S. Jou, "Load balanced Birkhoff-von Neumann switches: part I: one-stage buffering", *Comp. Commun.*, vol. 25, no. 6, pp. 611-622, Apr. 2002.
- [8] I. Keslassy, M. Kodialam, T.V. Lakshman and D. Siliadis, "On guaranteed smooth scheduling for input-queued switches", *IEEE INFOCOM 2003*, pp. 1384-1394, San Francisco, CA, Apr. 2003.
- [9] S. Li and N. Ansari, "Input-queued switching with QoS guarantees", *IEEE INFOCOM 1999*, pp. 1152-1159, New York, NY, Mar. 1999.
- [10] E.D. Jensen, C.D. Locke and H. Tokuda, "A time-driven scheduling model for real-time systems", *IEEE Real-Time Systems Symposium*, pp. 112-122, Dec. 1985.
- [11] D. Bertsekas, *Dynamic Programming and Optimal Control*, vol. 1 & 2, 2nd Ed., Athena Scientific, 2000.
- [12] A. Dua and N. Bambos, "Deadline aware scheduling for input queued packet switches", *Netlab Technical Report*, SU-Netlab-2005-06/02, Engg. Library, Stanford University
- [13] J. Walrand and P.Varaiya, *High-Performance Communication Networks*, 2nd Ed., Morgan Kaufmann, 1999.
- [14] K. Ross and N. Bambos, "Local search scheduling algorithms for maximum throughput in packet switches", *IEEE INFOCOM 2004*, pp. 1158-1169, Hong Kong, Mar. 2004.
- [15] S. Boyd and L. Vandenberghe, *Convex Optimization*, Cambridge University Press, 2003.