Local Search Scheduling Algorithms for Maximal Throughput in Packet Switches

Kevin Ross and Nicholas Bambos
Department of Management Science and Engineering
and Department of Electrical Engineering
Stanford University
Stanford, CA 94305-4026
Email: kross, bambos @stanford.edu

***Abstract*— We consider the (generalized) packet switch scheduling problem, where the switch service configuration has to be dynamically chosen based on observed queue backlogs, so as to maximize the throughput. A class of recently developed 'projective' scheduling algorithms, which substantially generalize the well-known maximum weight matching (MWM) algorithms for crossbar switches, are explored from the perspective of complexity. The typically huge number of possible switch configurations that the scheduler has to consider in each timeslot has been previously observed to lead to an impractical computational requirement.**

We introduce a new class of projective schedules based on 'local search' concepts. In particular, rather than searching the entire (typically huge) set of available service configurations to find the best one, the new scalable scheduling algorithms search 'locally' over a small neighborhood of service configurations to find a 'better' one in each time slot. We show that local projective scheduling algorithms can provide dramatic reduction in complexity *without* causing any loss of throughput (although they may observe higher delay). We explore the nature and structure of such schedules, which show a much higher promise for practical implementation than their global versions.

I. Introduction

We consider a generalized packet switch as a processing system having several queues, where $X_q(t)$ is the backlog (number of cells) in queue q at time slot t. In each time slot, the system can be set to a single service configuration S, chosen form the set $\mathcal{S}$ of all feasible ones. When the system is set to S in a time slot, S_q cells are removed from queue q in that slot. The objective is to dynamically choose and schedule the service configurations in consecutive time slots, so as to maximize the system throughput.

The most recognized example of this model is the crossbar packet switch. In this special case, each queue stores packets waiting to be sent between a particular input and output port pair. A packet arriving to the input port is stored in this virtual output queue until the switch is configured to connect it to its output port. Since each port can establish exactly one connection in each timeslot, the set of available configurations corresponds to the matchings of input and output pairs.

It has been known [12] that throughput maximization for this generalized switch system is achieved by a broad family of (global) *projective schedules*, which choose a service vector $S \in \mathcal{S}$ that maximizes over the whole $\mathcal{S}$ the inner product

$$\langle S, \mathbf{B}X \rangle = \sum_p \sum_q S_p B_{pq} X_q \tag{I.1}$$

(projection of S on $\mathbf{B}X$, hence, *projective* schedules), when the backlog vector is X, for any fixed matrix $\mathbf{B} = \{B_{pq}\}$ which is *positive-definite*, *symmetric*, and has *negative or zero off-diagonal elements*. These schedules substantially generalize the Maximum Weight Matching (MWM) algorithms [7], [10], [8], where $\mathbf{B}$ is simply the identity matrix. However, projective scheduling algorithms may be very computationally intensive and potentially impractical in some switching application scenarios, because the set $\mathcal{S}$ may have a huge number of elements S to calculate the inner product at.

In this paper, we introduce a substantial extension of projective schedules based on 'local search' ideas. Rather than searching the entire set of available service configurations $\mathcal{S}$ to find the one of maximal projection on $\mathbf{B}X$, the new scalable algorithms search over a much smaller 'local subset' of service configurations. This provides a large reduction in operational overhead, but these simple local schedules still achieve 100% of the throughput. What is actually traded for reduced complexity and overhead is a potentially increased delay.

In section II we describe the model, and in section II-B the standard projective schedules. In section III we introduce local projective schedules and prove that they also support maximum throughput. In section IV we explore how these policies apply to crossbar switch scheduling and explore their performance through simulation. We conclude with section V, and provide the technical proofs in appendices.

II. The Switching Model

Consider a queueing system comprised of Q queues, indexed by $q \in \mathcal{Q} = \{1, 2, ..., Q\}$. Time is slotted $t \in \{1, 2, 3, ...\}$. Let $A_q(t)$ be the (integer) number of cells arriving to queue q in timeslot t. We assume that arrivals to each queue q in a single timeslot are bounded above by some maximum burst ceiling $\overline{A}_q$. Arriving cells are buffered immediately in their respective queues, and served in a first-in-first-out (FIFO) manner[1] within each queue. Several consecutive cells may comprise a packet, but it is assumed that packets can be served preemptively[2] on a cell-by-cell basis. For each $q \in \mathcal{Q}$ we assume that

$$\lim_{t\to\infty} \frac{\sum_{s=0}^{t} A_q(s)}{t} = \rho_q \in (0, \infty) \qquad \text{(II.1)}$$

that is, the long-term average arrival load to each queue is well-defined, positive and finite. We make no further restriction on the traffic traces. In particular, we do not assume any particular statistics that may generate the traffic traces, or even independence between the individual queues.

The traffic vector during the time slot t is $A(t) = (A_1(t), A_2(t), ..., A_q(t), ..., A_Q(t))$ and the long-term average load vector is $\rho = (\rho_1, \rho_2, ..., \rho_q, ..., \rho_Q)$. We use vector notation throughout, referring to subscripts only when necessary.

At any time slot, the system may be set to a single service configuration S, chosen from a finite set of available configurations $\mathcal{S}$. Each configuration $S \in \mathcal{S}$ is a Q-dimensional vector

$$S = (S_1, S_2, ..., S_q, ..., S_Q) \qquad \text{(II.2)}$$

where S_q is the (integer) number of cells from queue q served (and removed from that queue) in a time slot, when the system operates under S. For example, if $Q = 3$, the service configuration $(3, 0, 4)$ refers to serving 3 cells from queue 1, 0 cells from queue 2, and 4 cells from queue 3 in a time slot when this configuration is used. Since configurations are defined by vectors we use the terms 'service configuration' and 'service vector' interchangeably.

In a standard **crossbar packet switch**, each input port may be connected to at most one output port to enable packet transfer in a time slot. An input-output port connectivity pattern can be represented by a service vector of 0's and 1's. For example, a 2-by-2 crossbar switch has four virtual output queues, buffering packets based on the arrival-destination port pairs $(1, 1), (1, 2), (2, 1)$ and $(2, 2)$. Respectively, the two main service configuration vectors of the switch are $(1, 0, 0, 1)$ and $(0, 1, 1, 0)$. This is illustrated in Figure 1.

The key design and operational concern under consideration is the dynamic scheduling of service configurations $S \in \mathcal{S}$, so as to maximize the system throughput. The backlog state $X(t)$ of the system is the Q-dimensional vector of cells in the individual queues

$$X(t) = (X_1(t), X_2(t), ..., X_q(t), ..., X_Q(t)) \qquad \text{(II.3)}$$

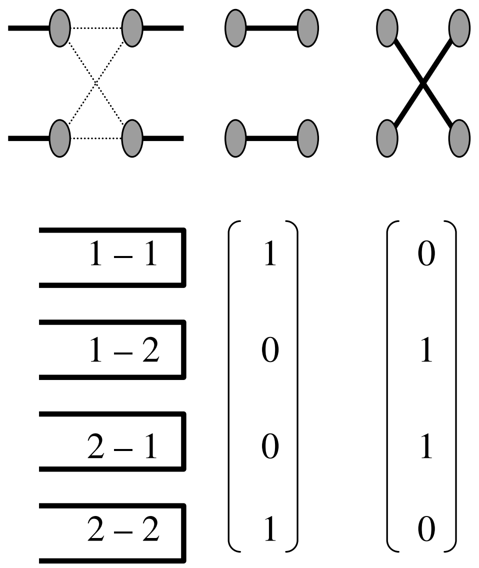
where $X_q(t)$ is total number of cells waiting in queue $q \in \mathcal{Q}$ at time t. The service state $S(t)$ of the queueing system at time $t > 0$ is the service vector

$$S(t) = (S_1(t), S_2(t), ..., S_q(t), ..., S_Q(t)) \qquad \text{(II.4)}$$

chosen from $\mathcal{S}$ by the scheduling algorithm (policy) that the system operates under. In general, $S(t)$ may depend on the observed history of the backlog and service states of the system.

[1]Actually, FIFO is not essential in the results that follow; it is merely a default service discipline.

[2]This requirement can be relaxed, but for clarity and simplicity of exposition we keep it here.



$$\begin{pmatrix}1\\0\\0\\1\end{pmatrix} \quad \begin{pmatrix}0\\1\\1\\0\end{pmatrix}$$

Fig. 1. The crossbar packet switch. The upper row shows the schematic of a standard 2-by-2 packet switch and the two input-output port connectivity configurations it can be set at. The lower row shows the four input-output virtual queues of the 2-by-2 switch and the two service vectors corresponding to the two connectivity configurations above. In general an N-by-N switch has N input and N output ports, with N^2 queues and $N!$ matching configurations.

If in time slot t the scheduling algorithm uses the service configuration $S(t) \in \mathcal{S}$, the number of cells departing from queue q is given by

$$D_q(t) = \min\{X_q(t), S_q(t)\} \qquad \text{(II.5)}$$

By convention, service is committed (and cell removals registered) at the beginning of each time slot, while cell arrivals are registered at the end. The backlog state of the system evolves according to the equation

$$X(t + 1) = X(t) + A(t) - D(t) \qquad \text{(II.6)}$$

or

$$X(t + 1) = X(0) + \sum_{z=1}^{t} A(z) - \sum_{z=1}^{t} D(z) \qquad \text{(II.7)}$$

where each term is a Q-dimensional integer vector. The objective in these systems is to select $S(t)$ (and hence $D(t)$), without any knowledge of future arrivals, in a way that ensures that all packets are served and no backlog queue is growing uncontrollably.

A. Stability and Throughput

We utilize the concept of rate stability in our throughput analysis of the system. In particular, we seek algorithms which ensure that the long-term cell departure rate from each queue is equal to the long-term arrival rate. Such algorithms must satisfy

$$\lim_{T\to\infty} \frac{\sum_{t=1}^{T} D_q(t)}{T} = \lim_{T\to\infty} \frac{\sum_{t=1}^{T} A_q(t)}{T} = \rho_q \qquad \text{(II.8)}$$

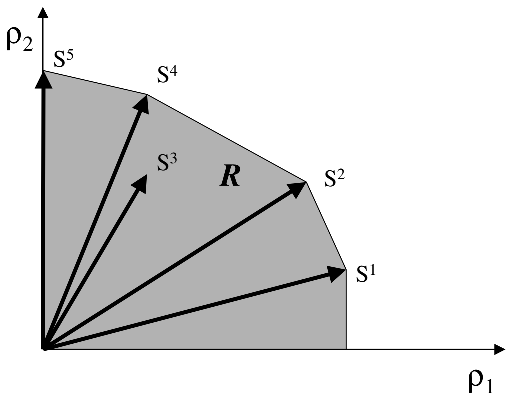


Fig. 2. The stability region. The set of allowable arrival rate vectors ρ is called the stability region $\mathcal{R}$. For any ρ in the region $\mathcal{R}$ above, there is a convex combination of service configurations which would apply a total service rate to each queue which is at least the arrival rate to that queue. For ρ outside there is no such combination.

for each $q \in \mathcal{Q}$, that is, there is cell *flow conservation* through the system. A scheduling algorithm which ensures II.8 is referred to as *rate stable*. It can be easily seen from II.7 and II.8 that

$$\lim_{t\to\infty} \frac{X(t)}{t} = 0 \tag{II.9}$$

is equivalent to rate stability.

The **stability region** $\mathcal{R}$ of the switching system described above is the set of all load vectors ρ for which rate stability (II.8) is maintained under at least one feasible scheduling algorithm. The stability region can be expressed [1], [2] as

$$\mathcal{R} = \{\rho \in \Re_+^Q : \ \rho \le \sum_{S\in\mathcal{S}} \phi_S S, \text{ for some } \phi_S \ge 0 \text{ with } \sum_{S\in\mathcal{S}} \phi_S \le 1\} \tag{II.10}$$

Intuitively speaking, a load vector ρ is in the stability region $\mathcal{R}$ if it is dominated (covered) by a convex combination of the service vectors $S \in \mathcal{S}$. This is illustrated in figure 2.

Indeed, if one knows ρ in advance, using each S for a fraction ϕ_S of time (in a TDM fashion) will ensure that equation II.8 is satisfied. On the contrary, if $\rho \notin \mathcal{R}$ it is impossible to maintain rate stability and flow conservation in all queues no matter what feasible schedule we use; hence, at least one queue will suffer an outflow deficit compared to the cell inflow.

An equivalent [12], [14] polar characterization of II.10 is given by

$$\mathcal{R} = \{\rho \in \Re_+^Q : \ \langle \rho, v\rangle \le \max_{S\in\mathcal{S}} \langle S, v\rangle \text{ for all } v \in \Re^Q\} \tag{II.11}$$

The intuition behind II.11 is that by shifting around the direction vector v we can 'scope out' the stability region, where ρ's projection on any v is smaller than that of some service vector $S \in \mathcal{S}$.

We are primarily interested in dynamic scheduling algorithms which maintain rate-stability and provide flow conservation (II.8) for all $\rho \in \mathcal{R}$, responding only to backlog and automatically adapting to ρ without any prior knowledge of it.

B. Projective Scheduling Algorithms

We consider a family of scheduling algorithms, which are called projective because they select the service vector $S^* \in \mathcal{S}$ of maximal projection on $\mathbf{B}X$, when the backlog state is X, as follows.

Definition 2.1: (Projective Schedule) Given an arbitrarily fixed Q by Q matrix $\mathbf{B}$, the **projective schedule** is defined as selecting a service vector from the set

$$\mathcal{S}^*(X) = \{S^* \in \mathcal{S} : \langle S^*, \mathbf{B}X\rangle = \max_{S\in\mathcal{S}} \langle S, \mathbf{B}X\rangle\} \tag{II.12}$$

when the backlog vector is X. This is simply the set of all service vectors of maximal projection onto the vector $\mathbf{B}X$. If there is more than one such vector, any one can be arbitrarily selected. Note that, in a time slot t where the backlog is $X(t)$, the service vector $S(t) \in \mathcal{S}^*(X(t))$ selected by the projective schedule (determined by $\mathbf{B}$) is such that

$$\langle S(t), \mathbf{B}X(t)\rangle = \max_{S\in\mathcal{S}} \langle S, \mathbf{B}X(t)\rangle\,, \tag{II.13}$$

or

$$\boxed{S(t) = \text{argmax}_{S\in\mathcal{S}} \langle S, \mathbf{B}X(t)\rangle} \tag{II.14}$$

maximizing the projection of $S(t)$ on $\mathbf{B}X(t)$.

The natural question is under what conditions this scheduling algorithm achieves maximal throughput. This is resolved by the following result.

Theorem 2.1 (Stabilizing Matrices): If the Q by Q matrix $\mathbf{B}$ is

(a) positive-definite, and
(b) symmetric, and has
(c) negative or zero off-diagonal elements,

then the projective schedule II.14 induced by $\mathbf{B}$ maintains rate-stability of the switch for every $\rho \in \mathcal{R}$ and, hence, maximizes its throughput.

Proof: The proof of this specific result is given in [12], [14]. The much more general proof of Theorem 3.2 presented later covers it also. ∎

The above family of projective schedules of maximal throughput is very large. Indeed, it is as large as that of matrices $\mathbf{B}$ that have the three required properties (a), (b), (c) identified above. The Q^2 elements of $\mathbf{B}$ can actually be chosen or manipulated [14] to provide differentiated quality of service (QoS, delay, jitter, etc.) to the various queues and induce desirable load-balancing effects across them. In every case, as long as the above three properties hold, the system is guaranteed to have maximal throughput. With the focus of this work on developing localized versions, we move the discussion of the selection of matrix $\mathbf{B}$ to [14] and future work.

It is interesting to note that in the very special case where

- $\mathbf{B}$ is the identity matrix $\mathbf{I}$ (obviously satisfying the above three properties) and

- the set $\mathcal{S}$ is comprised of the specific service vectors (with 0,1 elements) of a standard (crossbar) switch

the induced projective schedule simply reduces to the well-known **maximum weight matching (MWM)** algorithm [7]. Indeed, observe that when $\mathbf{B} = \mathbf{I}$ and each service vector $S \in \mathcal{S}$ has only 0 or 1 elements, the resulting inner product $\langle S, X\rangle = \sum_q S_q X_q$ reduces to the standard weighted (by the queue backlogs) matching, and the chosen S maximizes it over $\mathcal{S}$.

C. Complexity of Projective Schedules

Consider now the steps that a projective scheduling algorithm with fixed $\mathbf{B} = \{B_{pq}\}$ takes at the beginning of every single time slot t during system operation:

1) Observe the current backlog $X(t)$.
2) Compute the projection
$$\langle S, \mathbf{B}X(t)\rangle = \sum_{pq} S_p B_{pq} X_q(t) \qquad \text{(II.15)}$$
for every single S in $\mathcal{S}$, and choose an S^* that maximizes it over $\mathcal{S}$.
3) Apply the chosen S^* in the current time slot.

The problem is that the set $\mathcal{S}$ may have a huge number of service combinations S and, hence, the second step may be very computationally intensive and practically infeasible to perform in real time. This is indeed the case in many practical applications. For example, an N-by-N crossbar switch has $Q = N^2$ input-output queues, but the number of service combinations grows factorially in N.

In view of this level of prohibitive complexity, the question arises whether there exist lower complexity projective scheduling algorithms which (1) do achieve *maximal system throughput* but (2) do not search the whole (potentially huge) set $\mathcal{S}$ to select a service combination in each time slot, but a *much smaller subset* or some specially chosen neighborhood.

III. Local Projective Scheduling Algorithms

Fortunately, the above complexity problem can be often overcome using a natural 'local search' idea explained in this section. This leads to a new family of 'local' projective schedules, which is much larger than the one identified in the previous section. Every local schedule is of much lower complexity than its global version, yet it is shown here to also maximize the throughput. The tradeoff is between lower complexity and potentially higher average delay (congestion), yet the key property of maximal throughput remains intact.

First, we introduce a geometric cone representation of the projective schedules of the previous section. This reveals a topological (graph) structure that can be imposed on the set of service configurations $\mathcal{S}$ and introduces a natural concept of locality and neighborhood between the service vectors $\mathcal{S}$. This allows the definition of the new class of local projective schedules, every element of which is then proven to be throughput optimal.

A. Cone Geometry of Projective Schedules

It turns out that projective schedules can be naturally represented in an intuitive geometric way, using a specially defined **cone structure**, as follows. For each service vector $S \in \mathcal{S}$, let $\mathcal{C}_S$ be the set of backlog vectors X for which S is chosen under the projective schedule, given the fixed matrix $\mathbf{B}$; that is,
$$\mathcal{C}_S = \{X \in \mathbb{Z}^Q_{0+} : \ \langle S, \mathbf{B}X\rangle = \max_{S' \in \mathcal{S}} \langle S', \mathbf{B}X\rangle\} \qquad \text{(III.1)}$$
This is simply the set of backlog vectors X, on which S has the maximal projection amongst all other service vectors in $\mathcal{S}$.

Observe now that the projective schedule can now be geometrically defined as follows:
$$\boxed{\text{when } X \in \mathcal{C}_S, \text{ use the service vector } \ S} \qquad \text{(III.2)}$$
The set $\mathcal{C}_S$ is a geometric *cone* because $\langle S, \mathbf{B}X\rangle \leq \langle S', \mathbf{B}X\rangle$ implies that $\langle S, \mathbf{B}\alpha X\rangle \leq \langle S', \mathbf{B}\alpha X\rangle$ for any positive scalar $\alpha \in \Re_+$ and $S, S' \in \mathcal{S}$. Note that the cones $\{\mathcal{C}_S, S \in \mathcal{S}\}$ form a partition of the backlog space. Figure 3 shows the cone structure in a simple example of a two-queue system with three service configurations. In general, some cones may actually be degenerate and several cones may share common boundaries.

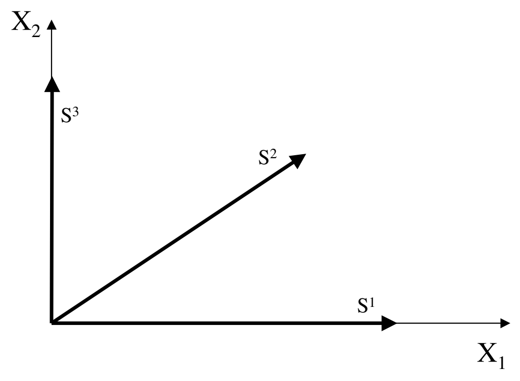
This cone structure is discussed in more depth [14], especially noting the effect of the matrix $\mathbf{B}$ on the boundaries of the cones. Since this work is focussed on the development of local projective algorithms, we use for examples the case where $\mathbf{B}$ is the identity matrix.

It is also useful (in the following proofs) to consider the cone $\mathcal{C}(X)$ of all backlog vectors that use the same service vector as X under the projective schedule (for a fixed $\mathbf{B}$). That is, if S is a service vector of maximal projection on X amongst all others in $\mathcal{S}$, then it is also of maximal projection on every other backlog vector $X' \in \mathcal{C}(X)$. Specifically,
$$\mathcal{C}(X) = \{X' \in \mathbb{Z}^Q_{0+} : \mathcal{S}^*(X') \subseteq \mathcal{S}^*(X)\} \qquad \text{(III.3)}$$
Note that, if X is in the interior of the non-degenerate cone $\mathcal{C}_S$, then $\mathcal{S}^*(X) = \{S\}$, so the only service vector that can be used by the projective schedule is S. However, if X is one the boundary of several adjacent cones, for example, $X \in \mathcal{C}_{S^1} \bigcap \mathcal{C}_{S^2} \bigcap \mathcal{C}_{S^3}$, then $\mathcal{S}^*(X) = \{S^1, S^2, S^3\}$ has multiple vectors and any of these can be used. That is why we have to use set inclusion in III.3.

B. Local Projective Schedules

Here we discuss how the cone geometry of section III-A leads to intuition in developing local projective schedules. In particular, we show that the scheduler need only consider a small subset of the available configurations at each timeslot. This property leads from the observation that the backlog X will usually only jump to a nearby service cone. This is illustrated in Figure 4 for a simple two-queue example, and discussed in more detail below.

Suppose that at (the beginning of) time slot t, the backlog vector is $X(t) = X$. We see from the evolution of the system



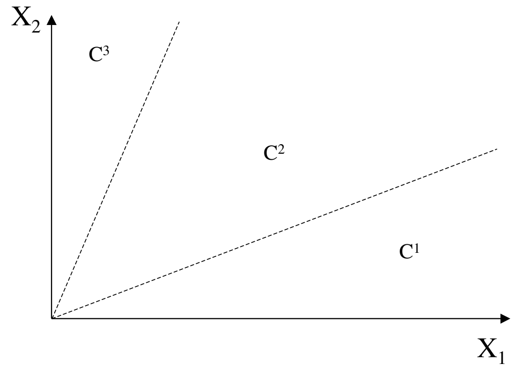


Fig. 3. The cone geometry of the projective schedule for a system of two queues ($Q = 2$) and three service vectors $S^1 = (4, 0), S^2 = (3, 2), S^3 = (0, 3)$. Note that S^1 serves queue 1 only, S^3 queue 2 only, and S^2 serves both in a somewhat balanced manner. The top graph shows the service configurations available. The bottom graph shows the corresponding cones $\mathcal{C}_S$ when the diagonal matrix $\mathbf{B} = [1, 0; 0, 1]$ is used; it would be different for a different $\mathbf{B}$. For example, if the backlog $X(t)$ is within the cone $C^1 = \mathcal{C}_{S^1}$ at time t then the projective schedule selects service configuration $S(t) = S^1$ at that time.

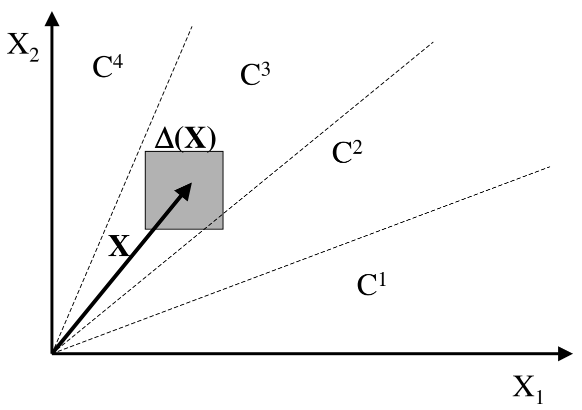


Fig. 4. *The intuition behind local projective schedules* demonstrated on a simple example of two queues and four service cones. The rectangle is the backlog displacement box defined in equation III.4. When the backlog is small the displacement box intersects many service cones and their service vectors have to all be considered to find the optimum. When the backlog is large, the box can at most intersect neighboring cones and only their service configuration need be considered.

in equation II.6 and the bounded arrivals and departures in a single timeslot that $X(t + 1)$ must be within some bounded distance from X. In particular, if we let $\bar{S}_q = \max\{S_q, S \in \mathcal{S}\}$ be the maximum number of cells that can be served in a single time slot from queue each $q \in \mathcal{Q}$ then we see that

$$X(t + 1) \in \Delta(X) = \{Y : X - \bar{S} \le Y \le X + \bar{A}\} \quad \text{(III.4)}$$

where $\Delta(X)$ is the maximum displacement 'box' of the backlog state $X(t + 1)$ in the next time slot. Note that the position of the displacement box $\Delta(X)$ depends on X, but its size does not. Extending the discussion in Figure 4, observe now the following:

1) When the backlog size $|X|$ is small, the displacement box $\Delta(X)$ within which $X(t+1)$ must lie may intersect many (or even all) service cones.
2) As $|X|$ gets larger, the displacement box $\Delta(X)$ (which does *not* grow in size) intersects fewer and fewer cones. Eventually, for large enough X it will intersect *at most* the *neighboring* cones of $\mathcal{C}_S$, when $X \in \mathcal{C}_S$.

The second observation is the critical one, given that we are primarily concerned with system stability, which is inherently a 'large load/backlog asymptotic' property. It reveals that the projective scheduler need only compute $\langle S^o, \mathbf{B}X(t + 1)\rangle$ for the small number of $S^o \in \mathcal{S}$ for which $\mathcal{C}_{S^o}$ is adjacent to $\mathcal{C}_S$, in order to identify the service vector $S(t+1)$ that maximizes it over all $\mathcal{S}$. This reduces complexity significantly and we formalize this idea below.

We first map the concept of *adjacent cones* to neighboring service vectors in $\mathcal{S}$, introducing a notion of locality and neighborhood in $\mathcal{S}$. Adjacent cones correspond to service cones with a common boundary. One way to view cone $\mathcal{C}_S$ is as the intersection of all backlog X half-spaces $\langle S, \mathbf{B}X\rangle \ge \langle S', \mathbf{B}X\rangle$ for all $S' \neq S$ and fixed S. The idea is to consider S and S' neighbors, if their cones $\mathcal{C}_S$ and $\mathcal{C}_{S'}$ are adjacent, sharing a unique non-degenerate boundary between them. Note that this construction does not include *trivial* neighboring cones which 'touch' at zero or along a hyper-plane corresponding to zero backlogs. More formally,

Definition 3.1: (Neighboring Service Configurations) Two distinct service vectors S and S' in $\mathcal{S}$ are **neighbors** if there exists a backlog $Y \ge 0$ for which

$$\langle S', \mathbf{B}Y\rangle = \langle S, \mathbf{B}Y\rangle > \langle S^o, \mathbf{B}Y\rangle \quad \text{(III.5)}$$

for all $S^o \in \mathcal{S}$ with $S^o \neq S, S'$. Define $\mathcal{N}(S)$ to be the set of neighbors of the service vector $S \in \mathcal{S}$. To attend to the non-typical case where multiple service vectors maximize the inner product - and are not neighbors according to baseline definition III.5 - we augment the set $\mathcal{N}(S)$ inductively as follows. If there exists a backlog $Z \ge 0$ and a service configuration $S^\diamond$ for which

$$\langle S^\diamond, \mathbf{B}Z\rangle > \langle S, \mathbf{B}Z\rangle > \langle S', \mathbf{B}Z\rangle \quad \text{(III.6)}$$

for all neighbors $S' \in \mathcal{N}(S)$ then $S^\diamond$ is added to $\mathcal{N}(S)$. These supplementary neighbors are defined iteratively and ensure that the boundaries of the cone $\mathcal{C}_S$ are completely represented by the neighbors $\mathcal{N}(S)$.

We can now define a *local* projective schedule such that, given that the service vector $S(t-1)$ was used in the $t-1$ time slot, the schedule will select $S(t)$ from the neighbor set $\mathcal{N}(S(t-1))$ (as opposed to the whole huge $\mathcal{S}$) in the next time slot, computed as follows.

Definition 3.2: (Local Projective Schedule) The local schedule is defined inductively, given a fixed matrix $\mathbf{B}$ and an arbitrary initial service vector $S(0)$. Let $S(t-1) \in \mathcal{S}$ be the service vector used in time slot $t-1$ and $X(t)$ the backlog vector at the beginning of time slot t, after service and arrivals in the previous time slot. Then, the **local projective schedule** selects a service vector $S(t) \in \mathcal{N}(S(t-1))$ to use in time slot t, such that

$$\langle S(t), \mathbf{B}X(t)\rangle = \max_{S \in \mathcal{N}(S(t-1))} \langle S, \mathbf{B}X(t)\rangle, \qquad \text{(III.7)}$$

that is,

$$\boxed{S(t) = \text{argmax}_{S \in \mathcal{N}(S(t-1))} \langle S, \mathbf{B}X(t)\rangle} \qquad \text{(III.8)}$$

This is similar to II.13 and II.14, except for the critical difference that only currently neighboring (local) service vectors in $\mathcal{N}(S(t-1))$ are considered in each step, as opposed to all available in $\mathcal{S}$, which generally reduces the computational complexity substantially.

Of course, the question that immediately arises is whether local projective schedules are rate stable for all loads $\rho \in \mathcal{R}$ and maximize throughput, or the reduction in complexity comes at a price of reduced throughput. It turns out that the throughput is not compromised at all, although the average delay may increase.

Theorem 3.1: (Stable Local Projective Schedules) If the Q by Q matrix $\mathbf{B}$ is (a) positive-definite, and (b) symmetric, and has (c) negative or zero off-diagonal elements, then the local projective schedule III.8 induced by $\mathbf{B}$ maintains rate-stability of the switch for every $\rho \in \mathcal{R}$ and, hence, maximizes its throughput.

Proof: The proof is covered by that of the following extended Theorem 3.2. ▮

We proceed to explore below an interesting graph-theoretic angle of local projective schedules, which reveals some important intuition about their nature and structure.

C. Local Search in Service Graphs - Drift Dynamics

It turns out that the dynamics of local projective schedules can be viewed as a walk on a specially structured graph with nodes corresponding to the service vectors. This walk is akin to 'gradient descent' dynamics, as explained below.

Consider the following *service graph* representation of the set $\mathcal{S}$. Each service vector $S \in \mathcal{S}$ forms a distinct node of the service graph; any two nodes S and S' are joined by an edge iff S, S' are neighbors according to Definition 3.1. The local projective schedule is then represented as a walk on this graph, as follows:

- Suppose the scheduler was on node S during the previous time slot (using service vector S) and (after cell arrivals and service throughout the slot) the backlog is X' at the beginning of the current time slot.
- The scheduler then computes $\langle S', \mathbf{B}X'\rangle$ on the neighboring nodes $S' \in \mathcal{N}(S)$ of node S on the service graph, and moves to the node where this inner product is maximized, picking the corresponding service vector to use in the current time slot.

The neighbors S' of S could be stored in a lookup table or a graph data structure. An example of a service graph is shown in Figure 5.

Let us now compare the global vs. the local projective schedules (given some fixed $\mathbf{B}$) and see how the latter drifts on the service graph being 'pulled' towards the former. Suppose the system was at node S in the previous time slot and at the beginning of the current one the backlog is X'. Now, the local scheduler scans the neighbors $\mathcal{N}(S)$ and picks $S' = \text{argmax}_{S^o \in \mathcal{N}(S)} \langle S^o, \mathbf{B}X'\rangle$ to move to. However, the global scheduler would scan the whole $\mathcal{S}$ and would pick $S^* = \text{argmax}_{S^o \in \mathcal{S}} \langle S^o, \mathbf{B}X'\rangle$. It is possible that S' is grossly suboptimal and very different from S^* being several tiers away from the latter on the service graph. This could easily happen if the backlog is small and a burst of arrivals pushes it to a cone which is well beyond the neighboring cones of the one that contained the original backlog.

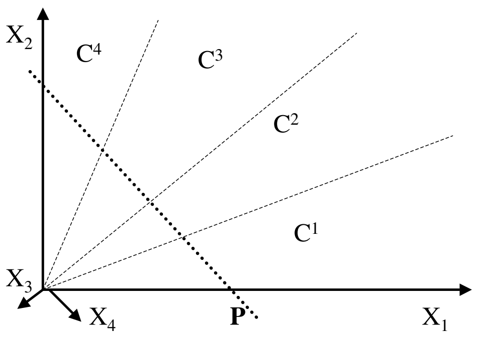
The lemma below reveals why the service choices of the local scheduler are always 'pulled' towards those of the global one and evolve towards the latter.

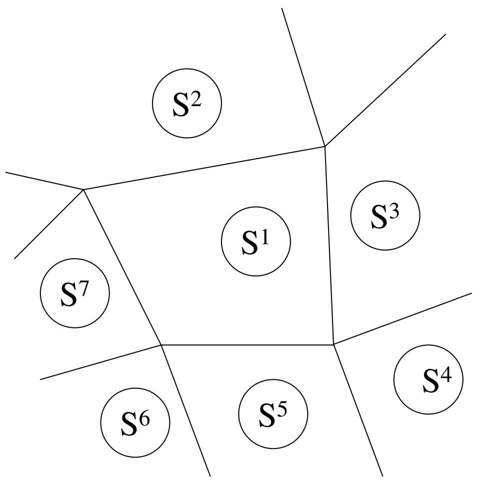
Lemma 3.1: (Local Drift towards Global Optimum) If the backlog is X and a globally sub-optimal service configuration $S \notin S^*(X)$ is used (i.e one that does not maximize the inner product $\langle S, \mathbf{B}X\rangle$ over $\mathcal{S}$), then there exists a neighbor S' of S for which $\langle S', \mathbf{B}X\rangle > \langle S, \mathbf{B}X\rangle$.

Proof: If X is such that $\langle S, \mathbf{B}X\rangle > \langle S', \mathbf{B}X\rangle$ for all neighbors $S' \in \mathcal{N}(S)$, then $\langle S, \mathbf{B}X\rangle > \langle S^o, \mathbf{B}X\rangle$ for all $S^o \in \mathcal{S}$ due to the definition of supplemental neighbors in equation III.6. But this would contradict the assertion that $\langle S^*, \mathbf{B}X\rangle > \langle S, \mathbf{B}X\rangle$ for $S^* \in S^*(X)$. ▮

Therefore, we can have **no local maxima**; there is only one service configuration that is better than all of its neighbors, the true optimal $S^* \in \mathcal{S}^*(X)$. Hence, in every time slot, if the current service is not optimal, it will always improve under the local scheduler. In other words, the local projective schedule 'pushes' the service toward the best configuration. The following corollary is now seen to be valid.

Corollary 3.1: If under the local projective schedule (given a fixed $\mathbf{B}$) the backlog $X(t)$ enters cone $\mathcal{C}_S$ (where the global schedule uses the optimal service vector S) at some time slot t_o and stays drifting in the same cone, it will eventually select S at most M time slots after entering the cone (where M is the number of available configurations in $\mathcal{S}$); after using S for the first time it will keep selecting it while in the cone. The pull to S may be much faster, as the worst case of M steps implies a neighborhood structure of only a single pair of neighbors to each cone.





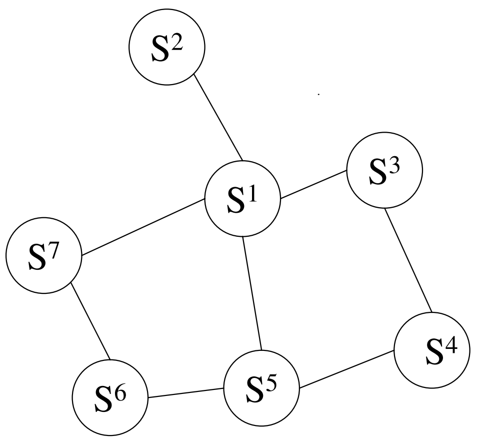


Fig. 5. *(Graph Representation of Local Search)* The local search method can be represented in graph form. The three figures above illustrate this. The first figure illustrates the division of the backlog space into cones as in figure 3. The second figure illustrates a cross section of this cone structure as projected onto a plane (such as P in the first graph, but in multiple dimensions). In general backlog cones may have a boundary in up to $Q-1$ dimensional space. In the third figure, the same cones as the second figure are considered. Each node corresponds to a service configuration and configurations are joined by an edge if they are neighbors. Each timeslot the previous configuration and its neighbors are considered. In this figure, if S^1 is used in timeslot $(t-1)$, then S^1, S^2, S^3, S^5 and S^7 would be considered in timeslot t.

D. *K-Delayed Projective Schedules*

The property of local projective schedules described in Corollary 3.1 is a special case of the more general property defined below, under which we prove the stability of an even larger family of relaxed projective schedules, extending even the local one discussed before.

Definition 3.3: (K-Delayed Projective Schedules) A projective schedule (given fixed $\mathbf{B}$) is defined to be **K-Delayed** if, whenever the system backlog $X(t)$ enters the cone $\mathcal{C}(Y)$ (for any fixed backlog Y) and stays there drifting for more than K slots, it will use $S^* \in S^*(Y)$ from the K^{th} time slot on, while in this cone.

It turns out that all K-delayed projective schedules (no matter what the integer K is) maximize the system throughput, under the following conditions.

Theorem 3.2: (Stable K-Delayed Projective Schedules) If the Q by Q matrix $\mathbf{B}$ is (a) positive-definite, (b) symmetric, and (c) has negative or zero off-diagonal elements, then any induced (by this matrix) K-delayed projective schedule (for any fixed K) is rate stable for any $\rho \in \mathcal{R}$ and, hence, maximizes the throughput.

Proof: The proof can be found in Appendix A. ∎

The local projective schedule is clearly K-Delayed for $K = M$ by corollary 3.1. This is only one of many scheduling algorithms which satisfy definition 3.3. The following are extensions or related algorithms which are also K-delayed projective schedules.

A. Instead of searching only immediate neighbors of the previous configuration, consider all nodes within r tiers. For example, 2nd tier neighbors are separated by two edges in the graph representation of figure 5.
B. Search neighbors until one is found which improves the previous $\langle S(t-1), \mathbf{B}X\rangle$, rather than continuing to check neighbors until the optimal one is found.
C. Consider just one (or generally l) service configuration in each time slot, and use $\arg\max\left\{\langle S(t-1), \mathbf{B}X\rangle, <\hat{S}, \mathbf{B}X>\right\}$ in such a way that $\hat{S}$ considers every configuration within some bounded number of time slots. For example, $\hat{S} = S^m, m = t(\mod M) + 1$ for an ordering $\{S^m\}_{m=1}^M$ of the service configurations.

The algorithms above are all K-delayed projective schedules for some finite integer K and, therefore maximize throughput as per Theorem 3.2. Scheme A is of higher complexity and may lead to lower average delay since the value of K will be lower, but this comes at the cost of greater computation. Scheme B is more efficient than standard local projective schedules since it is not required to search the entire neighborhood at each time, while scheme C is extremely simple to implement, since it checks only two configurations at each timeslot.

There is an intuitive relationship between the integer K and the performance of a switch. Since a high value of K indicates a longer lag before using the uptimal S, we can expect this to lead to a greater average buffer and delay for each queue.

This can be observed in the example application to the crossbar switch in figure 7 in the next section.

IV. Application to Crossbar Switches

Let us now consider how the algorithms described in Section III apply to the special, yet important, case of packet scheduling in crossbar packet switches. As was pointed out earlier (and has been explored in various other works), the set of service configurations available in an N-by-N crossbar packet switch corresponds to the set of matchings of size N.

We defined a neighbor relationship previously as a pair of service configurations for which there is a backlog X with inner product $\langle S, \mathbf{B}X\rangle$ maximized by only the two configurations. Now, we will consider the simpler case of $\mathbf{B} = \mathbf{I}$, the identity matrix, which corresponds to the special case of the maximum weight matching algorithm. It turns out that two matchings can be considered neighbors if the input-output ports can be loaded in such a way that the two matchings both serve equal maximum aggregate load.

It can be easily seen that there is a large number of neighbors for any particular configuration. Consider how one may check if two matchings are neighbors. For an arbitrary pair of matching configurations, if one cell is waiting in each of the queues served by S^1 and S^2 (including any queues in common), and no cells are waiting in any queue not served by these two configurations, then clearly S^1 and S^2 will maximize the inner product $\langle S, X\rangle$. The question remains as to when there is an X for which this maximization is *unique*.

We state here a constructive definition of neighbors, and then prove that these are in fact neighbors as defined earlier.

Consider an arbitrary pair of service configurations S^1 and S^2 in a switch. These can each be represented as a matching [7] (hence the name maximum weight matching). Now:

1) Overlay the matchings associated with S^1 and S^2 on a single graph $\mathcal{G}$.
2) Remove any single edges from $\mathcal{G}$ which correspond to common edges from both S^1 and S^2.

The following proposition reveals when two matchings are neighbors.

Proposition 4.1: If the two steps above lead to a graph $\mathcal{G}$ which is connected then S^1 and S^2 are neighbors. Otherwise they are not.

Proof: A constructive proof is given in Appendix B. ∎

Consider the example shown in Figure 6. This shows three matchings (configurations) of which the inner pairs are neighbors, but the outer pair are not.

In Figure 7 we plot the backlog trace under four different stable algorithms. The same arrival trace is used in each case. It can be seen that the local projective schedule is very close to the true global projective schedule (which in this case is equivalent to MWM). The 'first improved neighbor' algorithm is next best with the 'compare one configuration' also performing well. This progression of algorithms illustrates the tradeoff between complexity and quality of service. A lot less effort in computation, such as the 'compare one configuration' algorithm leads to an increase in buffer size and delay, but ultimately does not compromise the throughput.

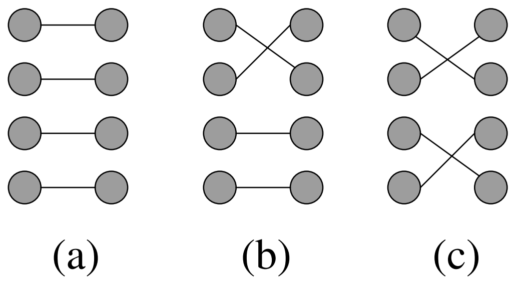


Fig. 6. **(Neighbors in a 4-by-4 packet switch)** . The above figure depicts 3 configurations in a 4-by-4 crossbar switch. Configuration pairs (a, b) and (b, c) are neighbors but (a) and (c) are not neighbors. They cannot be neighbors since, if the maximum matching were found with (a) and (c), then the matching for (b) would be at least as large, contradicting the definition of neighbors.

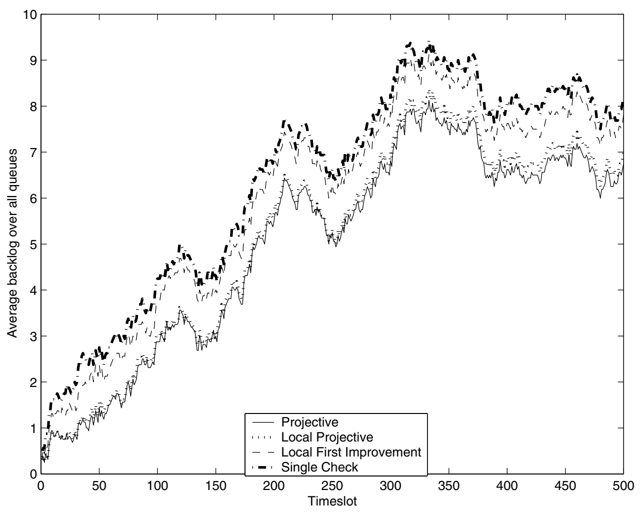


Fig. 7. *(Backlog Traces for a 4-by-4 switch)* . The same arrival sequence was applied to a 4-by-4 crossbar packet switch under four different algorithms. Arrivals were generated randomly from a poisson distribution at each timeslot with the arrival rate 99% of switch capacity. It can be seen that the local projective schedule performs almost as well as the global projective schedule with $\mathbf{B} = \mathbf{I}$. To reduce complexity even further, two other algorithms are illustrated. The local first-improvement scheduling algorithm checks the neighboring configurations until it finds an improvement on the previous configuration. The single check algorithm compares the previous configuration with one other configuration at each timeslot, cycling through all configurations over time. This progression of performance of the four algorithms illustrates the tradeoff between complexity and quality of service. The lower complexity algorithms lead to greater expected buffer and delay.

A. Further switch applications

The examples in this work have focussed on the crossbar packet switch application. The model itself, however, has much broader applications.

A switch with speedup available can be represented as vectors with the same queue structure. The service configurations available, however, include new vectors with elements of 0 and 2 instead of 0 and 1 (in the case where the speedup available is 2). This is a simple extension to allow the switch to serve two packets in a single timeslot. Because of the change in service configurations, the stability region expands to the convex combinations of the new configurations.

Multicast switches can also be described by the model described in this work. A multicast switch can connect each input port to multiple output ports, and arriving packets may be assigned multiple destination ports instead of a single one. Here we model each input to output set combination as a single virtual output queue, and the service configurations available describe the set of queues which can be served concurrently. In this application both the number of queues and the number of service configurations can grow exponentially, making the local projective schedules even more attractive.

Apart from the other switch architectures covered by the model, the K-delay projective schedules also show the throughput maximization of a number of useful extensions to the projective algorithms.

For example, consider using the projective schedule (global or local) except that one of the following conditions hold:

A. The switch is only able to change service configurations once every T timeslots instead of every timeslot.
B. The scheduler knows the number of waiting packets in each queue only approximately, within some finite bound of the actual backlog levels.
C. Arriving packets must be served consecutively without being broken into unit length cells. Here the packet length can be up to some finite bound.
D. The projective schedule solves equation II.13 within some bound rather than exactly. In this case,

$$\langle S(t), \mathbf{B}X(t)\rangle \geq \max_{S\in\mathcal{S}} \langle S, \mathbf{B}X(t)\rangle - E \qquad \text{(IV.1)}$$

for some bounded error E.

In each of these cases it can be shown that the schedule is a K-delay projective schedule (at least in the limit) for some finite K. These extended algorithms are extremely useful in practice as they correspond to realistic network and scheduling restrictions.

V. Conclusions and Further Research

We have shown that a large class of local projective algorithms lead to maximal throughput for the general switching model described in this paper. The prohibitive computation required for algorithms in the style of MWM can be reduced dramatically without any loss in throughput.

We have applied the general results to crossbar packet switching, as an important application. Simulation demonstrates a qualitative tradeoff between delay and complexity, and extended research will quantify this more clearly. Further work will also examine in depth the effect of the choice of matrix $\mathbf{B}$ and develop related algorithms.

Appendix

A. Proof of the Stability Theorem 3.2

In this appendix, we establish the proof regarding stability and maximal throughput of K-Delayed Projective Schedules, when the matrix $\mathbf{B}$ is positive-definite, symmetric, and has negative or zero off-diagonal elements. The proof method extends and covers corresponding ones in [2], [12].

First, note that the backlog 'drifts' between different cones as defined in equation III.1. This is illustrated in Figure 8. Recall that one key assumption in this proof is the K-delay property of the local schedule. That is, when the backlog enters a particular cone, the schedule selects the corresponding optimal service configuration (according to the global projective schedule) within a finite fixed time of K time slots and sticks with it while in this cone. This means that when the backlog enters a cone, there may be a time period of up to K time slots where a sub-optimal service vector is used, but after that the optimal one is picked up.

We consider an arbitrarily fixed arrival trace $A(t), t \in \{1, 2, 3, ...\}$ satisfying equation II.1 with $\rho \in \mathcal{R}$, where $\mathcal{R}$ is defined by equation II.10 or equivalently II.11. This infinite sequence of arrivals could be the result of a deterministic or probabilistic structure, and is assumed to be fixed.

The objective of this proof is to show that $\lim_{t\to\infty} \frac{X(t)}{t} = 0$. This is sufficient for rate stability by equation II.9. Since $\mathbf{B}$ is positive-definite it suffices to show that $\lim_{t\to\infty} \left\langle \frac{X(t)}{t}, \mathbf{B}\frac{X(t)}{t} \right\rangle = 0$.

Arguing by contradiction, let us assume that $\limsup_{t\to\infty} \left\langle \frac{X(t)}{t}, \mathbf{B}\frac{X(t)}{t} \right\rangle > 0$ and that this limit is attained on the increasing, unbounded time sequence $\{t_a\}_{a=1}^{\infty}$ with

$$\lim_{a\to\infty} \frac{X(t_a)}{t_a} = \eta \neq 0 \qquad (.1)$$

and the backlog blows up along[3] the vector direction η on $\{t_a\}_{a=1}^{\infty}$. Given this assumption, we establish a contradiction by showing there must exist a distinct (but related) time sequence $\{s_b\}_{b=1}^{\infty}$ which has the property

$$\lim_{b\to\infty} \left\langle \frac{X(s_b)}{s_b}, \mathbf{B}\frac{X(s_b)}{s_b} \right\rangle > \lim_{a\to\infty} \left\langle \frac{X(t_a)}{t_a}, \mathbf{B}\frac{X(t_a)}{t_a} \right\rangle = \langle \eta, \mathbf{B}\eta\rangle \qquad (.2)$$

which by definition should be the superior limit. The existence of the sequence $\{s_b\}_{b=1}^{\infty}$ contradicts the assertion that the superior limit is attained on $\{t_a\}_{a=1}^{\infty}$. We construct the series and identify four key properties, then show how these properties lead to the contradiction.

1) Constructing the sequence $\{s_b\}_{b=1}^{\infty}$: To construct the sequence $\{s_b\}_{b=1}^{\infty}$, we first construct a sequence $\{s_a\}_{a=1}^{\infty}$ related to $\{t_a\}_{a=1}^{\infty}$. The construction is based on the following intuition. The series $\{s_a\}_{a=1}^{\infty}$ correspond to times just prior to either (a) the backlog entering the cone $\mathcal{C}(\eta)$, or (b) a backlog queue becoming empty. This is made formal below. Let

$$r_a = \max\{t < t_a : X(t) \notin \mathcal{C}(\eta)\} \qquad (.3)$$

[3]Such a convergent subsequence should exist because $0 \leq \frac{X(t)}{t} \leq \frac{A(t)}{t} \to \rho \in \mathcal{R}$ which is a compact set.

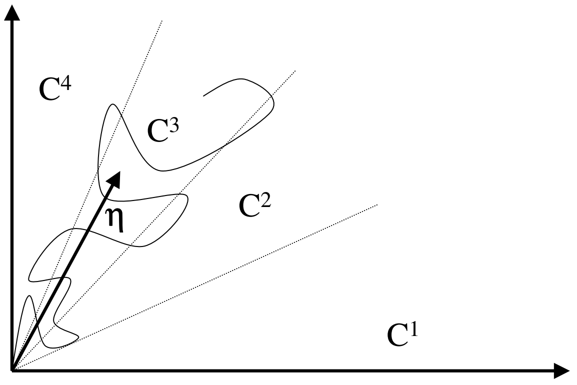


Fig. 8. The switch evolves with the backlog drifting within the cones that segment the backlog space. The contradiction basis of the proof assumes a limiting backlog direction vector η along which the backlog is linearly increasing over time (on a subsequence of time). This is shown by contradiction not to be possible, hence, the backlog-over-time ratio is eventually squeezed to zero and the switch is rate stable.

be the last time before t_a that the backlog $X(t)$ is not included in the cone $\mathcal{C}(\eta)$. This is the last time that $X(t)$ crosses from outside $\mathcal{C}(\eta)$ to inside, hence, $X(t) \in \mathcal{C}(\eta)$ for every $t \in [r_a+1, t_a]$ and the backlog drifts in $\mathcal{C}(\eta)$ throughout that interval. By convention $r_a = 0$ if the backlog has always been in $\mathcal{C}(\eta)$ before t_a.

Next we define another related sequence to assure that backlog queues are never empty in $[s_b+1, t_b]$. Let

$$q_a = \max\{t < t_a : X_q(t) = 0, \\ \text{for some } q \text{ with } \eta_q > 0\} \quad (.4)$$

We are now ready to construct sequence $\{s_a\}_{a=1}^{\infty}$. Let

$$s_a = \max\{r_a, q_a, (1-\epsilon_3)t_a\} \quad (.5)$$

for some $\epsilon_3 \in (0,1)$ and for all $a > 0$. The final term guards against the case where the backlog eventually remains in the same cone indefinitely.

Lemma 1.1: For $\{s_a\}_{a=1}^{\infty}$ and $\{t_a\}_{a=1}^{\infty}$ as constructed above, we have

$$\liminf_{a\to\infty} \frac{t_a - s_a}{t_a} = \epsilon > 0 \quad (.6)$$

We prove lemma 1.1 below, and given this result we select increasing, unbounded subsequences $\{s_b\}_{b=1}^{\infty}$ and $\{t_b\}_{b=1}^{\infty}$ of $\{s_a\}_{a=1}^{\infty}$ and $\{t_a\}_{a=1}^{\infty}$ on which the limit is equal to the inferior limit.

Thus we have established sequences for which

I. $\lim_{b\to\infty} \frac{t_b - s_b}{t_b} = \epsilon \in (0,1)$ and[4] $s_b < t_b$ for all b.

II. $X(t) \in \mathcal{C}(\eta)$ for all $t \in [s_b+1, t_b]$ and each b. This implies that the backlog $X(t)$ drifts within the cone surrounding η throughout the time interval $[s_b+1, t_b]$.

III. For each queue q such that $\eta_q > 0$, we have $X_q(t) > 0$ for all $t \in [s_b+1, t_b]$. There is no idle service to those queues in that time period.

IV. For each $t \in [s_b+K, t]$ we have $S(t) \in S^*(\eta)$. The service configuration is the globally optimal projective configuration from time[5] $s_b + K$ until t_b for some bounded K. This is related to the fact that the actual scheduler is K-delayed.

Properties II, III and IV follow directly from the definition of the series and the K-delay projective scheduler. It remains to prove lemma 1.1 which implies property I.

Proof: (of lemma 1.1): It suffices to show that the property is satisfied for the series $\{r_a\}_{a=1}^{\infty}$ and $\{q_a\}_{a=1}^{\infty}$. We first make the argument for $\{r_a\}_{a=1}^{\infty}$.

Arguing by contradiction, suppose that there exists an increasing unbounded subsequence $\{t_c\}_{c=1}^{\infty}$ of $\{t_a\}_{a=1}^{\infty}$ such that $\lim_{c\to\infty} \frac{t_c - r_c}{t_c} = 0$. Arrival bounds imply that $X(t_c) - X(r_c) \le (t_c - r_c)\overline{A}$. Dividing through by r_c and rearranging the algebra, we see that $\lim_{c\to\infty} \frac{X(t_c) - X(r_c)}{r_c} = 0$. Since $\lim_{c\to\infty} \frac{X(t_c)}{t_c} = \eta$, this implies $\lim_{c\to\infty} \frac{X(r_c)}{r_c} = \eta$. But according to this definition of r_c the backlog $X(r_c)$ must be outside $\mathcal{C}(\eta)$, so $\lim_{c\to\infty} \frac{X(r_c)}{r_c}$ could not converge to η. This establishes the necessary contradiction, showing $\liminf_{a\to\infty} \frac{t_a - r_a}{t_a} = \epsilon_1 > 0$.

For the series $\{q_a\}_{a=1}^{\infty}$ we consider the total arrivals between time q_a and t_a and the property that the backlog satisfies $\frac{X(t_a)}{t_a} \to \eta$. We see that $X_q \le (t_a - q_a)\overline{A}_q$. It follows that $\lim_{a\to\infty} \frac{t_a - q_a}{t_a} \overline{A}_q \ge \eta_q$ and hence $\liminf_{a\to\infty} \frac{t_a - q_a}{t_a} = \epsilon_2 > 0$.

The third term in equation .5 clearly satisfies the property. Since $\{s_a\}_{a=1}^{\infty}$ is defined as the termwise maximum of three sequences, all of which satisfy equation .6, the lemma is proved. ∎

2) Establishing the Contradiction: Here we establish how the properties of the constructed sequences lead to the aforementioned contradiction.

Lemma 1.2: If a sequence $\{s_b\}_{b=1}^{\infty}$ and a subsequence $\{t_b\}_{b=1}^{\infty}$ of $\{t_a\}_{a=1}^{\infty}$ satisfy the properties I to IV above then the superior limit is not attained on the sequence $\{t_a\}_{a=1}^{\infty}$ as initially assumed. This establishes the required contradiction.

Proof: Consider the change in backlog from time s_b to time t_b, that is,

$$X(t_b) - X(s_b) = \sum_{s_b}^{t_b - 1} (A(t) - D(t)) = \\ \sum_{s_b}^{s_b+K-1} (A(t) - D(t)) + \sum_{s_b+K}^{t_b-1} (A(t) - D(t)) \quad (.7)$$

Note that the first summation in the previous line is bounded and will be easily dealt with. The second is the interesting one. Note that, by property III, if $S_q(t) - D_q(t) > 0$ for any q and $s_b < t < t_b$, then $\eta_q = 0$. Hence we see that $[\mathbf{B}\eta]_q \le 0$ for all such q, due to the non-positive off-diagonal elements of $\mathbf{B}$, so $D_q[\mathbf{B}\eta]_q \ge S_q[\mathbf{B}\eta]_q$ for all q. Now projecting equation

[4]This implies that $\lim_{b\to\infty} \frac{s_b}{t_b} = 1 - \epsilon$.

[5]Note that from property I we eventually have $t_b > s_b + K$.

.7 onto $\mathbf{B}\eta$, we have

$$\langle X(t_b) - X(s_b), \mathbf{B}\eta\rangle \leq \sum_{s_b}^{s_b+K-1} \langle A(t) - D(t), \mathbf{B}\eta\rangle + \sum_{s_b+K}^{t_b} \langle A(t), \mathbf{B}\eta\rangle - \langle S^*, \mathbf{B}\eta\rangle (t_b - s_b - K) \quad (.8)$$

where $S^* \in S^*(\eta)$. The last term follows from properties II and IV, since the globally optimal service configuration S^* is used from from time $s_b + K$ until t_b. Finally, observe that

$$\lim_{b\to\infty} \frac{\sum_{s_b+K}^{t_b-1} A(t)}{t_b - s_b} = \lim_{b\to\infty} \frac{\sum_{s_b+K}^{t_b-1} A(t)}{t_b - s_b - K - 1} . \frac{t_b - s_b - K - 1}{t_b - s_b} = \rho \times 1 = \rho \quad (.9)$$

from property I[6]. Dividing equation .8 by $(t_b - s_b)$ and letting $b\to\infty$, we have

$$\lim_{b\to\infty} \left\langle \frac{X(t_b) - X(s_b)}{t_b - s_b}, \mathbf{B}\eta \right\rangle \leq \langle\rho, \mathbf{B}\eta\rangle - \langle S^*, \mathbf{B}\eta\rangle = -\gamma(\eta) \leq 0 \quad (.10)$$

from the equivalent definition II.11 of the stability region, setting $v = B\eta$.

Now, since $\{t_b\}_{b=1}^{\infty}$ is a subsequence of $\{t_a\}_{a=1}^{\infty}$ we have $\lim_{b\to\infty} \frac{X(t_b)}{t_b} = \eta$. Using property I and (.10) we get the following inequality

$$\lim_{b\to\infty} \left\langle \frac{X(s_b)}{s_b}, \mathbf{B}\eta \right\rangle = \lim_{b\to\infty} \left\{ \left\langle \frac{X(s_b) - X(t_b)}{s_b}, \mathbf{B}\eta \right\rangle + \left\langle \frac{X(t_b)}{s_b}, \mathbf{B}\eta \right\rangle \right\} = \lim_{b\to\infty} \left\{ \frac{s_b - t_b}{s_b} \left\langle \frac{X(t_b) - X(s_b)}{t_b - s_b}, \mathbf{B}\eta \right\rangle + \frac{t_b}{s_b} \left\langle \frac{X(t_b)}{t_b}, \mathbf{B}\eta \right\rangle \right\} \geq \frac{\epsilon}{1-\epsilon}\ \gamma(\eta) + \frac{1}{1-\epsilon}\ \langle\eta, \mathbf{B}\eta\rangle > \langle\eta, \mathbf{B}\eta\rangle \quad (.11)$$

The last inequality is due to the facts that $\epsilon \in (0,1)$ and $\gamma(\eta) \geq 0$. Now, by successive thinnings of the components of the backlog vector, we can obtain an increasing unbounded subsequence $\{s_c\}_{c=1}^{\infty}$ of $\{s_b\}_{b=1}^{\infty}$ such that

$$\lim_{c\to\infty} \frac{X(s_c)}{s_c} = \psi \quad (.12)$$

[6]For sequences $\{s_c\}_{c=1}^{\infty}$ and $\{t_c\}_{c=1}^{\infty}$ satisfying property I, we have

$$\lim_{c\to\infty} \frac{\sum_{s_c}^{t_c} A(t)}{t_c - s_c} = \lim_{c\to\infty} \frac{\sum_0^{t_c} A(t)}{t_c} \lim_{c\to\infty} \frac{t_c}{t_c - s_c} - \lim_{c\to\infty} \frac{\sum_0^{s_c} A(t)}{s_c} \lim_{c\to\infty} \frac{s_c}{t_c - s_c} = \rho\frac{1}{\epsilon} - \rho(\frac{1}{\epsilon} - 1) = \rho.$$

and from equation (.11)

$$\langle\psi, \mathbf{B}\eta\rangle > \langle\eta, \mathbf{B}\eta\rangle \quad (.13)$$

But $\mathbf{B}$ is *positive-definite* and *symmetric*, so equation .13 implies that $\langle\psi, \mathbf{B}\psi\rangle > \langle\eta, \mathbf{B}\eta\rangle$ or

$$\lim_{c\to\infty} \left\langle \frac{X(s_c)}{s_c}, \mathbf{B}\frac{X(s_c)}{s_c} \right\rangle = \langle\psi, \mathbf{B}\psi\rangle > \langle\eta, \mathbf{B}\eta\rangle = \limsup_{t\to\infty} \left\langle \frac{X(t)}{t}, \mathbf{B}\frac{X(t)}{t} \right\rangle \quad (.14)$$

giving a contradiction to the definition of η. This completes the proof of Lemma 1.2. ∎

Lemma 1.2, together with the construction in the previous section, completes the proof of Theorem 3.2.

B. Proof of Proposition 4.1

Here we prove the proposition concerning the neighborhood structure of the crossbar switch in two steps, as follows:

1) If the remaining graph $\mathcal{G}$ is connected then we construct a backlog vector X for which $\langle S, X\rangle$ is uniquely maximized at S^1 and S^2. Then by definition S^1 and S^2 are neighbors.
2) If $\mathcal{G}$ is not connected then we construct a third configuration S^3 which gives the same inner product $\left\langle S^1, X\right\rangle = \left\langle S^2, X\right\rangle = \left\langle S^3, X\right\rangle$ for any backlog X. Then, by definition, S^1 and S^2 are not neighbors.

Notice first that $\mathcal{G}$ is a graph of degree two. Each node has one connection from each matching. Further, each connected subset of $\mathcal{G}$ is a connected bipartite graph of degree two, hence has a unique Hamiltonian cycle. Within any connected bipartite graph of degree two one can choose exactly two matchings by following the Hamiltonian cycle and choosing odd or even edges.

Since each queue corresponds to a pair of ports, we replace the queue subscript q with the corresponding (i, j) pair of input and output ports.

First, we deal with the connected case. Let X be the backlog vector where one cell is waiting in each queue served by S^1 or S^2.

$$X_{ij} = \begin{cases} 1 & \text{if } S_{ij}^1 = 1 \text{ or } S_{ij}^2 = 1 \\ 0 & \text{otherwise} \end{cases} \quad (.15)$$

Clearly $\left\langle S^1, X\right\rangle = \left\langle S^2, X\right\rangle = N$. We consider any other S^3 with $\left\langle S^3, X\right\rangle = N$ (the upper bound by construction) and show that this must be equal to either S^1 or S^2. Note that S^3 must be constructed entirely of connections also in S^1 or S^2. If it any connection is different, $\left\langle S^1, X\right\rangle < N$ since one port is not serving any cell. This means that for every input port in S^3 it must be connected to its corresponding output port from either S^1 or S^2. Hence to construct S^3 we must first include the common elements of S^1 and S^2. Secondly, must choose a matching from the remaining edges represented in $\mathcal{G}$. Since $\mathcal{G}$ is a connected bipartite graph of degree two, there are only two matchings which can be chosen. These must correspond exactly to S^1 and S^2 and hence S^3 is not a different configuration. Therefore we have constructed a backlog X for which $\langle S, X\rangle$ is uniquely maximized by S^1 and S^2.

Next, we prove that if $\mathcal{G}$ is not connected then we can construct such an S^3 with $\big\langle S^3, X\big\rangle = \big\langle S^1, X\big\rangle = \big\langle S^2, X\big\rangle$ for any backlog vector X. Consider an arbitrary backlog vector X for which the inner product is maximized by both S^1 and S^2. First include the common edges in S^3. That is, $S^3_{ij} = 1$ if $S^1_{ij} = S^2_{ij} = 1$.

Since $\mathcal{G}$ is not connected and each node is of degree two, it can be separated into two disjoint graphs $\mathcal{G}_1$ and $\mathcal{G}_2$ each of which is also a graph of degree two. Let S^3 be constructed from the edges of $\mathcal{G}_1$ associated with S^1 and the edges of $\mathcal{G}^2$ associated with S^2. Clearly S^3 is different from S^1 and S^2 since otherwise S^1 and S^2 must be equivalent. Further, the inner product $\big\langle S^3, X\big\rangle$ must be equal to $\big\langle S^1, X\big\rangle$ and $\big\langle S^2, X\big\rangle$ since otherwise we have two cases:

1) If $\big\langle S^3, X\big\rangle > \big\langle S^1, X\big\rangle$ then this contradicts the assertion that S^1 and S^2 maximize the inner product.
2) If $\big\langle S^3, X\big\rangle < \big\langle S^1, X\big\rangle$ then consider S^4 which chooses the common edges plus the edges of $\mathcal{G}_1$ associated with S^2 and the edges of $\mathcal{G}^2$ associated with S^1. Now since the edges in S^3 and S^4 exactly correspond to those in S^1 and S^2 we have $\big\langle S^3, X\big\rangle + \big\langle S^4, X\big\rangle = \big\langle S^1, X\big\rangle + \big\langle S^2, X\big\rangle$ hence we must have $\big\langle S^4, X\big\rangle > \big\langle S^1, X\big\rangle$, again contradicting the assertion that S^1 and S^2 maximize the inner product.

Hence we can construct S^3 which is different from S^1 and S^2 but has at the same inner product value. Therefore S^1 and S^2 cannot be neighbors. ∎

REFERENCES

[1] Armony, M. and Bambos, N. (1999) Queueing networks with interacting service resources. Proceedings of 1999 Allerton Conference, Urbana, IL, pp. 42-51, 1999.

[2] Armony, M. and Bambos, N. (2001) Queueing dynamics and maximal throughput scheduling in switched processing systems. TR NetLab-2001-09/01, Engineering Library, Stanford University, 2001. To appear in Queueing Systems in 2003.

[3] Baccelli, F. and Bremaud, P. (1994) Elements of Queueing Theory, Springer-Verlag, 1994.

[4] Bambos, N. and Walrand, J. (1993) Scheduling and stability aspects of a general class of parallel processing systems. Advances in Applied Probability, 25:176-202, 1993.

[5] Kumar, P.R. and Meyn, S.P. (1995) Stability of queueing networks and scheduling policies, IEEE Transactions on Automatic Control, 40(2):251-260, 1995.

[6] Keslassy, I. and McKeown, N. (2001) Analysis of scheduling algorithms that provide 100% throughput in input-queued switches. Computer Systems Lab Report, Stanford University, 2001.

[7] McKeown, N., Mekkittikul, A., Anantharam, V., Walrand, J. (1999) Achieving 100% throughput in an input-queued switch. IEEE Transactions on Communications, 47(8):1260-1267, 1999.

[8] Stolyar, A. (2001) MaxWeight scheduling in a generalized switch: state space collapse and equivalent workload minimization under complete resource pooling. Preprint, 2001.

[9] Tassiulas, L. and Ephremides, A. (1992) Stability properties of constrained queueing systems and scheduling policies for maximum throughput in multihop radio networks. *IEEE Transactions on Automatic Control*, 37(12):1936-1948, 1992.

[10] Tassiulas, L. (1998) Linear complexity algorithms for maximum throughput in radio networks and input queued switches, IEEE INFOCOM'98, pp. 533-539, 1998.

[11] Tassiulas, L. and Bhattacharya, P. P. (1999) Allocation of interdependent resources for maximal throughput. *Stochastic Models*, 16(1).

[12] Ross, K. and Bambos,N. (2002) Projective Cone Schedules in Queueing Structures; Geometry of Packet Scheduling in Commnunication Network Switches, Proceedings of the 2002 Allerton Conference on Communication, Control and Computing. Monticello, Illinois.

[13] Young, D. Iterative Solution of Large Linear Systems, Academic Press: 1971

[14] Ross, K. and Bambos, N. (2002) Projective Processing Schedules in Queueing Structures; Applications to Packet Scheduling in Communication Network Switches, Technical Report SU NETLAB-2002-05/01, Engineering Library, Stanford University.