Forty-Seventh Annual Allerton Conference
Allerton House, UIUC, Illinois, USA
September 30 - October 2, 2009

Is it enough to drain the heaviest bottlenecks?

Gagan Raj Gupta
grgupta@purdue.edu

Sujay Sanghavi
sanghavi@mail.utexas.edu

Ness Shroff
shroff@ece.osu.edu

***Abstract*—This paper takes a philosophically new approach to throughput-optimal scheduling queueing systems with interference. All existing popular approaches (e.g. max-weight, greedy, ”pick-and-compare” etc.) focus on the weights of *individual* queues. We take an alternative approach, by focusing instead on the *aggregate queues of bottlenecks*. A bottleneck is a set of mutually-interfering queues; a schedule drains a bottleneck if it removes a packet from any one of its queues.**

We consider (the standard) switch scheduling problem, where the bottlenecks are the nodes. We establish the following phase-transition *(1)* ensuring *only* that the very heaviest nodes are drained is *not* enough for throughput optimality, but *(2)* ensuring scheduling for all nodes with weight within $(1-\alpha)$ of the heaviest *is* enough for throughput optimality, for any $\alpha > 0$. The proof uses a new Lyapunov function: the weight of the critical bottleneck.

Our alternate node-focused view also enables the development of new algorithms for scheduling. We show *(a)* how any policy can be made throughput-optimal by doing a small number of extra operations, *(b)* a new algorithm – Maximum Vertex-weighted Matching (MVM) – has (empirical) delay performance better than the current state of the art, and lower complexity than Max-(edge)weighted Matching, and *(c)* a class o f throughput-optimal policies that trade off between complexity and delay.

I. Introduction

Popular approaches in the theory and practice of scheduling in the presence of interference focus on determining schedules based on the weights, i.e. queue lengths, of individual links; examples include the ”max-weight” rule and its variants, greedy maximal scheduling, randomized algorithms based on the ”pick and compare” idea etc. This paper advocates the exploration of an alternate viewpoint, by focusing instead on the aggregate queues of system *bottlenecks*.

A bottleneck is a set of queues such that the service of any one of them precludes the simultaneous service of another. In terms of the ”conflict graph” of the queues, a bottleneck represents a maximal clique. This paper is based on the intuition that if we can drain, i.e. decrease the aggregate number of packets in, the heaviest / critical bottlenecks, then we should be able to obtain throughput optimality. We consider the first, simple case of scheduling in switches. At a high level, we show that

- Schedules determined based on the weights of the *ports / nodes* (where weight = total queue at that port) have several appealing algorithmic and delay properties.
- Analysis involves a new Lyapunov function: the weight of the heaviest bottleneck.

This work was supported in part by ARO W911NF-08-1-0238 and NSF award 07221236-CNS.

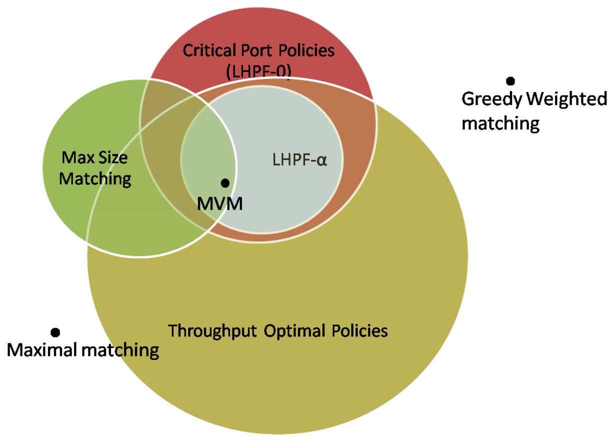

Fig. 1. Class of scheduling policies for switches.

We show a peculiar phase transition behavior: while scheduling all the critical bottlenecks, with highest weight, is *not* enough for throughput optimality, scheduling a maximal subset of the nodes that are within $(1-\alpha)$ of the heaviest *is* throughput optimal, for any $\alpha > 0$. We define a class of polices – LHPF-α – that are all throughput optimal, and can trade-off between throughput and delay based on the choice of α. Figure 1 summarizes the performance of our, and other known policies.

A. Related Work

In this paper we restrict our attention to the development of *online* algorithms, which attempt to schedule traffic by computing a matching every time slot. One such policy is the famous MWM policy which computes the maximum weight matching and is known to be throughput optimal. The proof for stability can be provided either in the fluid limits [2] or in the stochastic sense [18]. But essentially it hinges on a quadratic Lyapunov function and ensuring that the drift is negative.

The Maximum Size Matching (MSM) policy schedules the maximum size matching and hence maximizes the instantaneous throughput in each time slot. However it is known that if ties are broken randomly, MSM does not achieve 100% throughput for all admissible Bernoulli traffic patterns [11], [13]. It is possible that if the ties are broken carefully, a special MSM might be stable. Among the class of MSM policies, there are two polices that have been proposed in the literature to be throughput optimal: MVM and MWM-0+.

MVM is known to be throughput optimal [14]. The proof of throughput optimality in [14] uses the fact that a MVM on a graph G is a MWM on a graph $G^{'}$, where edge weights have been selected carefully. The technique to prove throughput optimality of MVM is essentially the same as that for MWM. The proof provided in this paper serves as a alternate, since MVM is a member of the LHPF-α class of policies.

MWM-0+ : At each time slot, consider all matchings which have maximal size. Among these choose one which has maximum weight, with weight function log. Break ties arbitrarily. This is conjectured to be throughput optimal in [16].

It is useful to also consider online scheduling according to maximal matches, which are matchings where no new edges can be added without sharing a node with an already matched edge. Maximal matchings can be found with O(N^2) operations and the computation is easily parallelizable to O(N) complexity [19]. Greedy weighted maximal matching (GMM) is a scheduler that tries to schedule the heavy edges. The GMM policy has been analyzed for the general class of networks with interference constraints [4] where it is shown that they achieve full throughput in a network that satisfies the local pooling condition. In simple terms, the local pooling condition means that a vector λ in the capacity region cannot dominate another vector μ in the capacity region in all the coordinates. This result can be generalized [1], [7], [8] to show that GMM achieves at least a certain fraction of the capacity region given by the local pooling factor. Although our Lyapunov function looks similar to that in [1], [4], [7], [8] it is based on node weights as opposed to weights on the individual edges in the graph. Moreover, we can show that the LHPF-α class of policies are not even required to be maximal in every time-slot whereas the policies considered in [1], [7], [8] are.

As noted in [14], the MVM policy combines the benefit a maximum size algorithm, with those of a maximum weight algorithm, while lending itself to simple implementation in hardware. In MVM, each weight is a function of queue lengths (sum of all edges that touch a node) and hence it has an advantage of both the maximum size matchings with high instantaneous throughput while guaranteeing high throughput, even when the arrival traffic is non-uniform. We have in fact characterized a class of policies much larger than the MVM policy and potentially lower complexity and equivalent performance benefits.

II. Preliminaries

Switches: This paper considers scheduling in (the standard) input-buffered crossbar switches, which we now briefly describe. An $N_1 \times N_2$ input-buffered crossbar switch contains N_1 input ports and N_2 output ports. The system operates in discrete time slots. In each slot, packets may arrive at the input ports; each packet has an output port it needs to be transferred to. Packets have to be transferred from inputs to outputs, under the following constraint: in any one time slot each input port can send at most one packet to at most one output port, and each output port can receive at most one packet from at most one input port. The scheduling problem is to determine how to transfer packets subject to these constraints.

Notation: Switch scheduling can be modeled as the problem of finding matchings in bipartite graphs, one in every time slot. Consider $G(s)$ the graph at slot s. $G(s)$ is a bipartite graph with input ports on one side and output ports on the other. As mentioned in the introduction, we will use "nodes" and "ports" interchangeably. There is an edge (i, j) in $G(s)$ if and only if there is at least one packet at input i that has output j as its destination. The scheduling algorithm finds a matching $M(s)$ in $G(s)$; then, for every edge $(i, j) \in M(s)$ one packet is then transferred from i to j. These packets are then considered to have left the system. A scheduling policy is a rule to pick the matching $M(s)$, in every slot s, based on the state of the system. For any input port i, $q_i(s)$ denotes the *total* number of packets at i. Similarly, for any output port j, $q_j(s)$ denotes the total number of packets in the system (i.e. all inputs) that are waiting to be transferred to j. We will not need to refer to the queues on individual edges. We will however often refer to the total queue at a port as the "weight" of that port; "heavy" ports have more packets in their queues than "lighter" ports.

We now state a couple of well-known results, from [6], [12], [15] which we will use in the proofs of this paper.

Lemma 1 (Hall's Condition): Let G be any bipartite graph, with the two partitions being V_1 and V_2. Let $S_1 \subset V_1$ be any subset of one partition. Then, there exists a matching in G that matches every node in S_1 if and only if for every further subset $S \subset S_1$, we have that $|\mathcal{N}(S)| \geq |S|$. Here the neighborhood $\mathcal{N}(S)$ is all nodes in V_2 that have an edge to some node in S.

Lemma 2: Let G be any bipartite graph, with the two partitions being V_1 and V_2. Let $S_1 \subset V_1$, and suppose there exists a matching M_1 that matches all nodes in S_1. Similarly, let $S_2 \subset V_2$ and there exist and M_2 that matches all nodes in S_2. Then there exists a matching M that matches all nodes in *both* S_1 and S_2.

Note that in Lemma 2, M may not match the nodes in the two sets to each other; just that each node in $S_1 \cup S_2$ will be matched to some node in the graph.

Graph-theoretic preliminaries:
We now formally define the terms we will use. All are standard, except for the definition of "absorbing paths". Throughout, we consider a **node-weighted graph.** The *length* of a path is the number of edges it contains. The *weight $w(M)$ of a matching M* is the total weight of all the nodes it matches. For any two matchings M_1 and M_2, the *symmetric difference*, denoted by $M_1 \triangle M_2$, is the set of edges in one of the two matchings, but not in both. It is well known that $M_1 \triangle M_2$ is always the node-disjoint union of paths and even-length cycles. Finally, given a matching M and path P, the set $M \oplus P = M - (M \cap P) + (M^c \cap P)$ denotes the edges obtained by "flipping" the edges in P. We now define the two scenarios of our interest where the resulting set $M \oplus P$ is also a matching.

Given a matching M, and any node i not matched by M,

1) An *augmenting path from unmatched node i* is any odd-length path P whose every alternate edge is in M, has

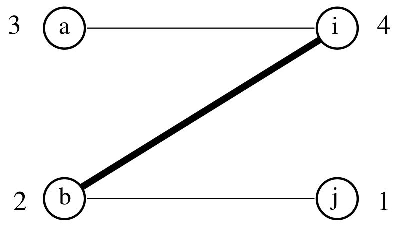

Fig. 2. Augmenting and absorbing paths. Edge (b, i) is in the matching M. $a - i - b$ is an absorbing path. $a - i - b - j$ is an augmenting path.

i as one endpoint, and ends at an unmatched node (say j).

2) An *absorbing path from unmatched node i* is any even-length path P whose every alternate edge is in M, has i as one endpoint, and whose last endpoint – say j – has weight $w_j < w_i$.

Note: $w(M \oplus P) > w(M)$ for any matching M and any augmenting or absorbing path P. Fig. 2 illustrates the idea of augmenting and absorbing paths. $a - i - b$ is an absorbing path from a since it is an even-length path ending in a node with smaller weight. $a - i - b - j$ is an augmenting path from a since it is a odd-length path and ends in an unmatched node j.

III. LHPF-α Policies

We now define the class of policies – Lazy Heaviest Port First (LHPF) -α – that we will be considering, and investigate algorithmic implementations. The next section proves their throughput-optimality (our main result).

Definition: Let $h = \max_i q_i$ be the weight of the heaviest port. Any port i whose weight satisfies $q_i \in [(1 - \alpha)h, h]$ is an *α-heavy port*. A matching M is LHPF-α if it schedules as many of the α-heavy nodes as possible, with heavier nodes taking strict priority over lighter ones. A policy is LHPF-α if in every time slot it picks an LHPF-α matching in every time slot.

The main result of this paper is that any LHPF-α policy is throughput-optimal, for any $\alpha > 0$. This is stated and proved in Section IV. Additional properties (elaborations for each follow):

(1) $\alpha = 0$ corresponds to **critical port policies**: those that only guarantee the scheduling of the very heaviest ports, with weight equal to h. These can *evacuate* optimally, but are not troughput-optimal.

(2) Any LHPF-α matching (and hence policy) is also a LHPF-β matching (policy) for $\beta < \alpha$.

(3) $\alpha = 1$ corresponds to the **MVM** – Maximum Vertex-weighted Matching – policy. This has lower worst-case complexity than the popular Max-(edge)weight matching, and simulations show that it *also* has lower delays.

(4) Note that for $\alpha < 1$, the policy **may not even be maximal**. In fact small α can be expected to result in very few nodes being scheduled most of the time. This is why we call our class of policies ”lazy”.

(5) α trades off between complexity and delay, while remaining throughput-optimal. Lower α corresponds to lower complexity, but higher delays.

(6) Augmenting and absorbing paths are key to forming LHPF-α matchings in node-weighted graphs, just as augmenting paths are for edge-weighted matchings.

(7) Any matching (and hence any policy) can be **post-processed** to become LHPF-α. This allows us to convert low-complexity but non-throughput-optimal policies into throughput-optimal ones via few extra operations.

A. Algorithms

Just as algorithms for edge-weighted matchings are based on augmenting paths, those for node weighted matchings are based on alternating *and* augmenting paths:

Lemma 3: M is an LHPF-α if and only if none of the unmatched α-heavy nodes has an augmenting or absorbing path.

This immediately leads to an algorithm to **convert a given (possibly empty) matching into an LHPF-α matching.**

Procedure for LHPF-α

```
INPUT: a node-weighted graph, and any initial matching $M_0$ (which could be empty, or generated by some other algorithm)
OUTPUT: $M^*$, an LHPF-$\alpha$ matching
• Set $l = 1$
• At iteration $l$,
    – IF $M_{l-1}$ matches all heavy nodes, set $M^* = M_{l-1}$ and BREAK.
    – Pick any unmatched heavy node $i$ in $M_{l-1}$, and try to find an augmenting or absorbing path $P$ from $i$.
    – IF such a $P$ can be found, set $M_l = M_{l-1} \oplus P$ and $l = l + 1$.
    – ELSE set $M^* = M_{l-1}$, and BREAK loop.
```

Remarks: The above description is just a conceptual procedure; efficient implementations could potentially rely on optimizations (e.g. like parallelism, as was done in [9] for max-cardinality matching). We emphasize, rather, a more interesting aspect of the above procedure: is that it allows us to make LHPF-α matchings out of non-LHPF ones via **post-processing.** In particular, the initial matching M_0 could be generated by a very simple algorithm (e.g. maximal matching, or edge-based greedy); the above algorithm would then convert it into LHPF-α. Lemma 3 implies that LHPF defines a nested class of matchings / policies.

Lemma 4: Let M be a LHPF-α matching. Then for any $\beta < \alpha$, M is also an LHPF-β matching.

For clarity, we now describe a policy that is *not* an LHPF-α policy. Suppose we do the following: go down the sequence of ports, recursively matching nodes if any neighbor is free, but *not* changing the edges already previously matched. Even if we go all the way to the end, this policy is not LHPF-α because it may exclude a port that would have been possible to schedule by changing the matchings of heavier ports that came before it. It is thus important that the ports are added via augmenting or absorbing paths.

B. Maximum Vertex-weighted Matching

$\alpha = 1$ corresponds to Maximum Vertex-weighted Matching (MVM) problem: find the matching with the highest node weight.

Lemma 5: M is an MVM if and only if there is no augmenting path or absorbing path from any of its unmatched nodes.

The complexity of MVM is $O(N^{2.5}log(N))$ [17] and the policy is simple to implement in hardware [14]. Many heuristics have been developed for MSM and they can be readily tuned to compute approximate MVMs. with the characterization of LHPF-α policies, which is a much bigger class, we expect that it would be much easier to develop heuristics for LHPF-α matchings.

C. Critical Port Policies

A port i is a *critical port* if it has the heaviest weight – i.e. $q_i = h = \max_j q_j$.

Lemma 6: In any switch configuration (i.e. where node weights are total queues at the ports), all the critical ports can always be *simultaneously* matched.

The proof follows from the Birkhoff-VonNeumann theorem. This leads us to consider the class of *critical port policies*: a policy is a critical port policy if it schedules every critical port in every time slot.

Lemma 7 (Optimal Evacuation): Consider a switch with an initial loading and *no further arrivals*. Then, a policy evacuates the system (i.e. tranfers every packet) in minimum time *if and only if* it is a critical port policy.

While evacuation-time optimality is a very appealing short-term property, critical port policies do not have good long-term performance. In particular, they are *not throughput optimal*. To see this, consider a 2x2 switch with the sequence of arrivals shown in Table I, where the scheduler schedules only the critical ports. a_{ij} represents the number of packets arriving at link (i, j) at that time, and q_{ij} represents the queue lengths under a policy that *only* schedules critical ports. As can be seen from the table, if the same arrival pattern continues, the value of q_{11} and q_{22} will keep on increasing with time and will eventually become unbounded. Note that the average load in the system is within the capacity region and the arrival sequences also satisfies the law of large numbers. Hence we conclude that scheduling only the critical ports is not enough. In general, to achieve throughput optimality, we have to schedule "enough" heavy ports.

TABLE I
CRITICAL-PORT POLICIES ARE NOT THROUGHPUT OPTIMAL.

T	$a_{11}, a_{12}, a_{21}, a_{22}$	$q_{11}, q_{12}, q_{21}, q_{22}$
0	0, 0, 0, 0	0, 0, 0, 0
1	0, 0, 0, 2	0, 0, 0, 2
2	2, 0, 0, 0	2, 0, 0, 1
3	1, 0, 0, 0	2, 0, 0, 1
4	0, 0, 0, 1	1, 0, 0, 2
5	0, 0, 0, 2	1, 0, 0, 3
6	2, 0, 0, 0	3, 0, 0, 2
7	1, 0, 0, 0	3, 0, 0, 2
8	0, 0, 0, 1	2, 0, 0, 3
9	0, 0, 0, 2	2, 0, 0, 4
10	2, 0, 0, 0	4, 0, 0, 3
11	1, 0, 0, 0	4, 0, 0, 3
12	0, 0, 0, 1	3, 0, 0, 4
13	0, 0, 0, 2	3, 0, 0, 5
14	2, 0, 0, 0	5, 0, 0, 4
15	1, 0, 0, 0	5, 0, 0, 4
16	0, 0, 0, 2	4, 0, 0, 6

IV. THROUGHPUT OPTIMALITY OF LHPF-α POLICIES

Theorem 1: Any LHPF-α policy is throughput optimal, for any $\alpha > 0$.
Proof:

Let the system be empty at time 0. Let $a_i(n)$ denote the cumulative number of packets that have arrived at an input port i up to time slot n. Similarly, $a_j(n)$ denotes the cumulative number of packets that have arrived in the system, destined for output port j up to time slot n. For each edge in the matching, one packet is removed at both the nodes touching the edge. With this understanding, henceforth, we shall not distinguish between an output and input port. We assume the convention that $a_i(0) = 0$. We assume that the arrival processes $a_i(.)$ satisfy a strong law of large numbers (SLLN): with probability one,

$$\lim_{n\to\infty} \frac{a_i(n)}{n} = \lambda_i \qquad (1)$$

For any port, input or output, let λ_i be the average rate of arrival of packets to port i. Define

$$\epsilon^* \;=\; \min_i (1 - \lambda_i)$$

The capacity region is $\{\lambda : \lambda_i < 1 \text{ for all } i\}$, which means that $\epsilon^* > 0$.

Fluid Model

We develop a fluid limit model following the development in [2]. Let $q_i(n)$ denote the weight at port i and $d_i(n)$ be the number of packets that departed from port i by time slot n. Let $h_M(n)$ be the number of slots in which matching $M \in \mathcal{M}$ has been scheduled, where $\mathcal{M}$ is the set of all matchings (not necessarily maximal). Then h_M is a non-decreasing function. Also note that by definition of $G(n)$, M can schedule only non-zero edges in the system. M_i indicates if matching M schedules port i. Note that $q_i(.)$ and $d_i(.)$ evolve according to the following:

$$q_i(n) = q_i(0) + a_i(n) - d_i(n)$$

$$d_i(n) = \sum_{M\in\mathcal{M}} \sum_{l=1}^{n} M_i(h_M(l) - h_M(l-1))$$

$$\sum_{M\in\mathcal{M}} h_M(n) = n$$

We define $a_i(t)$ for a non-negative real number t by interpolating the value of a_i between time $\lfloor t \rfloor$ and $\lfloor t \rfloor + 1$. We also define $q_i(t)$ and $d_i(t)$ in the same way by linear interpolation of the corresponding values at time $\lfloor t \rfloor$ and $\lfloor t \rfloor + 1$. Then, by using the techniques of Theorem 4.1 of [3], we can show that, for almost all sample paths and for all positive sequence $x_k \to \infty$, there exists a subsequence x_{k_l} with $x_{k_l} \to \infty$ such that the following convergence holds uniformly over compact intervals of time t:

For all i,

$$\frac{a_i(x_{k_l}t)}{x_{k_l}t} \to \lambda_i t \qquad \frac{q_i(x_{k_l}t)}{x_{k_l}} \to Q_i(t) \tag{2}$$

$$\frac{d_i(x_{k_l}t)}{x_{k_l}} \to D_i(t) \qquad \frac{h_i(x_{k_l}t)}{x_{k_l}} \to H_i(t) \tag{3}$$

$$\tag{4}$$

The system (D, H, Q) is called the fluid limit and queues evolve in the fluid limit as follows:

$$Q_i(t) = Q_i(0) + \lambda_i(t) - D_i(t)$$

$$\frac{d}{dt}D_i(t) = \sum_{M \in \mathcal{M}} M_i \frac{d}{dt}H_M(t)$$

$$\sum_{M \in \mathcal{M}} H_M(t) = t$$

D, H and Q are absolutely continuous functions and are differentiable at almost all times $t \geq 0$ (called *regular* times). It follows that

$$\begin{aligned} \frac{d}{dt}Q_i(t) &= \lambda_i - \frac{d}{dt}D_i(t) \\ &= \lambda_i - \sum_{M \in \mathcal{M}} M_i \frac{d}{dt}H_M(t) \end{aligned} \tag{5}$$

The following lemma from [2] establishes the connection between the stability of the switch and the fluid model.

Lemma 8: A switch operating under a matching algorithm is rate stable if the corresponding fluid model is weakly stable.

Lemma 9: The fluid model of a switch operating under a matching algorithm is weakly stable if for every fluid model solution D, T, Q with Q(0)=0, Q(t)=0 for almost all $t \geq 0$.

Define Lyapunov function

$$V(Q(t)) \;=\; \max_i Q_i(t)$$

Note that in the definition of V the maximum is taken over *all* ports, input and output.

Remarks

- The Lyapunov function used by [4] for the analysis of GMM policy also looks at the maximum queue length. The novelty of our proof is that we do not need to look at the individual queue lengths. Our Lyapunov function is based on port weights. Another difference is that while the analysis in [4] depends on the fact that GMM is a maximal matching, our proof works for all LHPF-α policies which are not even required to be maximal, in general.
- Our proof of stability is more subtle than the proof of stability for the MWM policy [2]. Note that the maximum weight matching in the graph remains the maximum weight matching in the corresponding fluid model. However, the ports that are critical in a given interval of time $(t, t+\delta)$ in the fluid model may not be critical on a slot by slot basis in the actual system. Hence, for example, a critical port policy may not be able to schedule all the ports that are critical in the fluid model.

Proof Intuition

Our proof is based on the observation that all the ports that are critical (heaviest) in the fluid limit, may not remain heaviest in the neighborhood of time t, but they continue to be above a certain threshold. We show that all ports that are critical in the fluid limit are scheduled in *every* time-slot around t by any LHPF-α policy for $\alpha > 0$. We prove this in Lemma 10. Note that the LHPF-α policy does not need to know which ports are critical in the fluid limit.

Theorem 2: Any LHPF-α policy is throughput optimal.

Proof:

Since V(Q(t)) is a non-negative function, to show that $V(t) = 0$ for almost all $t \geq 0$, it is enough to show that, if t is a regular time and $V(t) > 0$ then V(Q(t)) decreases at least at a given rate.

We prove that for all regular times t such that $V(Q(t)) > 0$, for a system operating under any LHPF-α policy for $\alpha > 0$,

$$\frac{d}{dt}V(Q(t)) \leq -\epsilon^*$$

Fix time t and let $\gamma = V(Q(t)) = \max_i Q_i(t)$. Also, define

$$\mathcal{C} \;=\; \{i : Q_i(t) = \gamma\}$$

to be the set of heaviest ports at t. Also, let $\widetilde{\gamma} = \max_{i \notin \mathcal{C}} Q_i(t)$ be the heaviest of the remaining ports. Since the number of ports is finite, $\widetilde{\gamma} < \gamma$. Choose β small enough so that *(a)* $\widetilde{\gamma} < \gamma - 3\beta$, and *(b)* $\beta < min\{\frac{\gamma\alpha}{2}, \frac{\gamma}{2N+1}\}$. Here $N = max\{N_1, N_2\}$. Note that this implies that

$$(\gamma - \beta) \;>\; (\gamma + \beta)(1 - \alpha) \tag{6}$$

and also

$$\left(\frac{N+1}{N}\right)(\gamma - \beta) \;>\; (\gamma + \beta) \tag{7}$$

Recall that $Q(t)$ is absolutely continuous. This means that there exists a δ small enough, so that at all times $\tau \in (t, t+\delta)$ the queues satisfy the following conditions

(C1) $Q_i(\tau) \in (\gamma - \frac{\beta}{2}, \gamma + \frac{\beta}{2})$ for all $i \in \mathcal{C}$

(C2) $Q_i(\tau) < \gamma - \frac{5\beta}{2}$ for all $i \notin \mathcal{C}$

Let x_{k_l} be a positive subsequence for which the convergence to the fluid limit holds. Consider l large enough so that $|\frac{q_i(x_{k_l}t)}{x_{k_l}} - Q_i(t)| < \frac{\beta}{2}$.

Consider time slots $T := \{\lceil x_{k_l}t \rceil, \lceil x_{k_l}t \rceil + 1, \ldots \lfloor x_{k_l}(t+\delta) \rfloor\}$. The following lemma shows that all critical ports that are critical at the fixed time t in the fluid limit will be scheduled at all time slots $n \in T$. The conditions (C1) and (C2) can be rewritten as follows for the original switching system.

(C1*) $q_i(n) \in [x_{k_l}(\gamma-\beta), x_{k_l}(\gamma+\beta)]$ for all $i \in \mathcal{C}$

(C2*) $q_i(n) \leq x_{k_l}(\gamma-2\beta)$ for all $i \notin \mathcal{C}$

We state a lemma. We prove it immediately after the current proof.

Lemma 10: For all times $n \in T$, any LHPF-α policy with $\alpha > 0$ will match all ports that are in $\mathcal{C}$ at time t in the fluid limit.

Now, assuming that a LHPF-α policy indeed schedules every port $i \in \mathcal{C}$ at all times $n \in T$,

$$\sum_{M\in\mathcal{M}} M_i(h_M(\lfloor x_{k_l}(t+\delta)\rfloor)-h_M(\lceil x_{k_l}t\rceil)) = \lfloor x_{k_l}(t+\delta)\rfloor-\lceil x_{k_l}t\rceil \tag{8}$$

Now by dividing both sides by x_{k_l} and let $l \to \infty$, we obtain:

$$\begin{aligned} 1 &\geq \frac{\sum_{M\in\mathcal{M}} M_i(h_M(x_{k_l}(t+\delta)) - h_M(x_{k_l}t)}{x_{k_l\delta}} \\ &\geq \frac{\lfloor x_{k_l}(t+\delta)\rfloor - \lceil x_{k_l}t\rceil}{x_{k_l\delta}} \to 1 \end{aligned} \tag{9}$$

Hence for $\delta \to 0$,

$$\begin{aligned} \sum_{M\in\mathcal{M}} M_i \frac{d}{dt} H_M(t) &= \lim_{\delta\to 0} \sum_{M\in\mathcal{M}} M_i \frac{H_M(t+\delta) - H_M(t)}{\delta} \\ &= \lim_{\delta\to 0} \lim_{l\to\infty} \frac{\sum_{M\in\mathcal{M}} M_i(h_M(x_{k_l}(t+\delta)) - h_M(x_{k_l}t)}{x_{k_l}(\delta)} \\ &\to 1 \qquad \text{by Eq. (9)} \end{aligned} \tag{10}$$

So, by Eq. (5) it follows that, $\forall i \in \mathcal{C}$,

$$\frac{dQ_i(t)}{dt} = -(1-\lambda_i) \leq -\epsilon^*. \tag{11}$$

Also, every port $i \notin \mathcal{C}$ has weight strictly lower than every port in $\mathcal{C}$, for the entire duration $(t, t+\delta)$. Thus it follows that

$$\frac{d}{dt}V(Q(t)) \;\leq\; -\epsilon^*$$

This proves the theorem. ■

Proof of Lemma 10:

We first note that all the ports in $\mathcal{C}$ are *heavy ports* for the LHPF-α policy on account of conditions (C1*) and Eq. (6). Let $\mathcal{C}_1 \subset \mathcal{C}$ be the set of input ports in $\mathcal{C}$, and $\mathcal{C}_2 \subset \mathcal{C}$ the set of output ports. We will first show that all ports in $\mathcal{C}_1$ can be matched, by showing that Hall's condition (given in Lemma 1) holds for this set. By symmetry, all ports in $\mathcal{C}_2$ can be matched and by Lemma 2 we conclude that all ports in $\mathcal{C}$ can be matched.

Fix time $n \in T$, and for any subset $S \subset \mathcal{C}_1$ let $\mathcal{N}_\tau(S)$ be its neighborhood at time n. Suppose now that S fails Hall's condition, i.e. that $|S| \geq |\mathcal{N}_n(S)| + 1$. Now, each $i \in S$ has $q_i(n) > x_{k_l}(\gamma-\beta)$, by condition (C1). This means that

$$\sum_{i\in S} q_i(n) \;>\; |S| x_{k_l}(\gamma-\beta) \;\geq\; (|\mathcal{N}_n(S)|+1)\, x_{k_l}(\gamma-\beta)$$

Now, each packet in $q_i(n)$, $i \in S$ is destined for one node in $\mathcal{N}_n(S)$, which means that

$$\sum_{j\in\mathcal{N}_n(S)} q_j(n) \;\geq\; \sum_{i\in S} q_i(n)$$

(LHS and RHS may not be equal because there might be other input ports with packets for ports in $\mathcal{N}_n(S)$). This means that there exists one node $j^* \in \mathcal{N}_n(S)$ with queue

$$\begin{aligned} q_{j^*}(n) &\geq \frac{1}{|\mathcal{N}_n(S)|} \sum_{j\in\mathcal{N}_n(S)} q_j(n) \\ &> \left(\frac{|\mathcal{N}_n(S)|+1}{|\mathcal{N}_n(S)|}\right) x_{k_l}(\gamma-\beta) \end{aligned}$$

Further, $|\mathcal{N}_n(S)| \leq N$, so we have that

$$\frac{|\mathcal{N}_n(S)|+1}{|\mathcal{N}_n(S)|} \;\geq\; \frac{N+1}{N}$$

which gives, from (7),

$$q_{j^*}(n) \;>\; \left(\frac{N+1}{N}\right) x_{k_l}(\gamma-\beta) > x_{k_l}(\gamma+\beta)$$

However, this means that j^* violates the fact, implied by (C1*) and (C2*), that $q_j(n) < x_{k_l}(\gamma+\beta)$ for all ports j. This is a contradiction, and thus it has to be that Hall's condition is satisfied at n.

Thus, there exists a matching that matches all input ports in $\mathcal{C}_1$. Similarly, it can be shown there exists a matching that matches all output ports in $\mathcal{C}_2$. By Lemma 2, this means there exists a matching that matches all ports in $\mathcal{C}$. From conditions (C1*) and (C2*), it follows that this matching matches all ports with weight greater than $\gamma-\beta$.

Since $\beta < \gamma\alpha$, all the ports with weight above $\gamma-\beta$, i.e. all ports in $\mathcal{C}$ will be simultaneously scheduled by the LHPF-α policy. ■

V. Simulations

In this section we study the performance of LHPF-α policies as a function of α. We also study the performance of the LHPF-α policies obtained by post-processing a maximal schedule and compare its delay performance with MWM-α algorithms. MVM lies in the class of LHPF-α policies.

We implemented a packet level simulator in Java. The simulations are run long enough so that the half-width of the 99% confidence interval is within 1% of the mean.

We simulate a 8×8 switch with symmetric loading on each edge. We simulate two types of arrival processes, Bernoulli and a more bursty arrival process. Each arrival stream injects packets independently in the system. Clearly, these processes satisfy strong law of large numbers and the switch is guaranteed to be stable. The model of the bursty arrival process is described below.

Bursty Arrival Processes: The arrival stream is a series of active and idle periods. During the active periods, the source injects one packet into the queue in every time slot. The length of the active periods (denoted by random variable a) are distributed according the Zipf law with power exponent

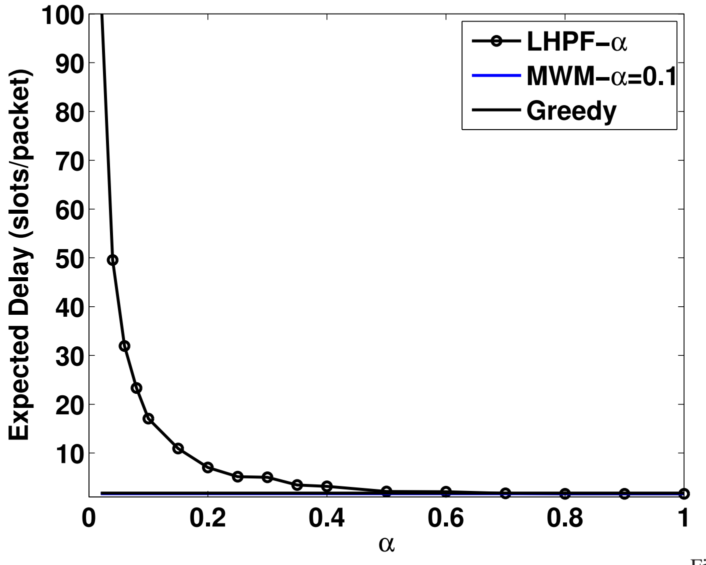

Fig. 3. Average Delay in a 8 × 8 switch under LHPF-α policy under symmetric Bernoulli traffic at system utilzation $\rho = 0.45$.

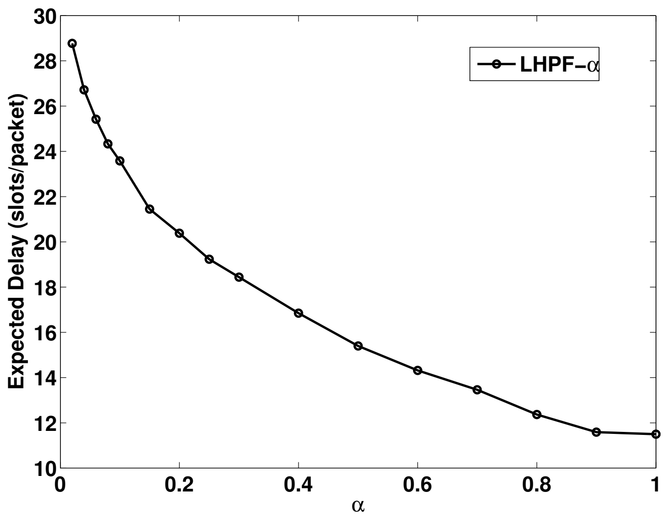

Fig. 5. Delay of a 8 × 8 switch under post-processed maximal matching policy (with LHPF-α) under symmetric Bernoulli traffic at system utilzation $\rho = 0.95$.

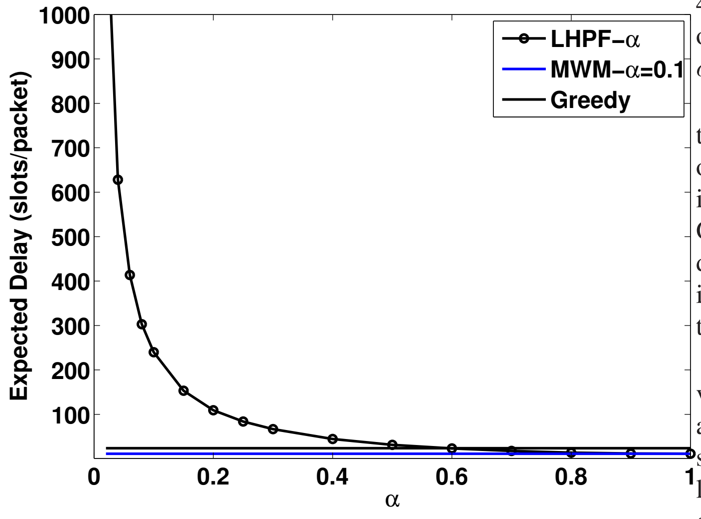

Fig. 4. Delay of a 8 × 8 switch under LHPF-α policy under symmetric Bernoulli traffic at system utilzation $\rho = 0.95$.

1.25 and support [1,2,3,...,100]. Heavy tailed distributions like Zipf, have been found to model the Internet traffic [5]. During the active period the source generates one packet every time-slot. The idle periods are geometrically distributed with mean p. The mean arrival rate of a source can be controlled by changing the value of p.

We first study the performance of the LHPF-α policies as a function of α. In this study, we generate the LHPF-α matchings from scratch, i.e. we post-process an empty matching and schedule only from the set of *heavy nodes* for the particular value of α. The average delay of packets in the system for symmetric Bernoulli traffic is shown in Figs. 3 and 4 for a system utlization ρ, 0.45 and 0.95 respectively. We observe that the delay is inversely proportional to the value of α and is especially large for small values of α.

This can be thought of as a trade-off between the computational complexity of the scheduler and the delay performance of the system. The computational complexity of the scheduler is as small as O($N^{2.5}$) for $\alpha < 1/N$ while it is as large as O($N^{2.5} logN$) for $\alpha = 1$. While small value of α leads to large delays, the delay is much smaller for $\alpha = 1$ at the cost of increased computational overheads per slot. We also observe that the delay increases with system utilization ρ.

Post-processing provides an interesting alternate to circumvent the above trade-off between computational complexity and delay performance. It is well known from the simulation studies that in most cases, computationally simple algorithms like maximal matching, greedy maximal matchings etc. have good delay performance. However, they are not throughput optimal and thus can have bad delay performance in some cases. By post-processing a matching and making it LHPF-α, we can guarantee throughput optimality with small computational overhead.

We study the delay performance of LHPF-α matchings obtained by post-processing maximal matchings. The average delay of packets in the system as a function of α for symmetric Bernoulli traffic is shown in Fig. 5. By comparing the results in Fig. 5 and 4, we conclude that post-processing offers significant gains both in complexity and in delay performance.

We compare the performance of LHPF-α policies with that of MWM-α policies for Bernoulli traffic. The delay performance of LHPF-($\alpha = 0.3$) is similar to that of the MWM policy for significantly less computational overhead. The value of α serves as a tunable parameter that increases the delay efficiency at the cost of increased computational

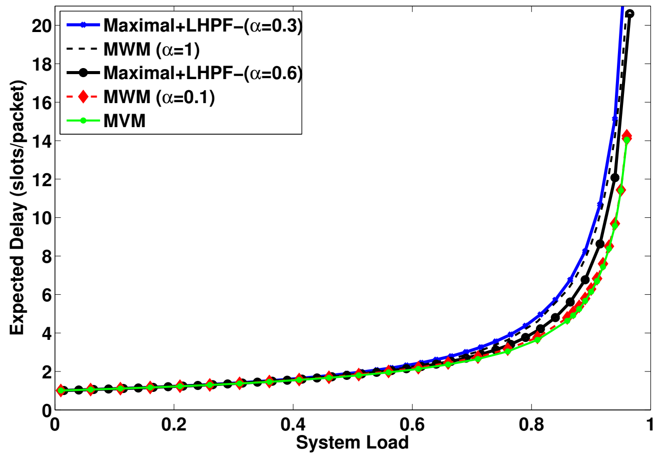

Fig. 6. Delay of a 8 × 8 switch under symmetric Bernoulli traffic.

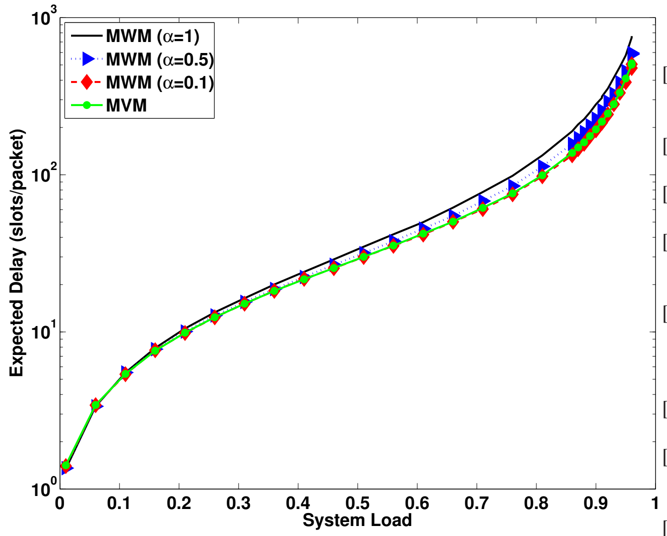

Fig. 7. Delay of a 8 × 8 switch under symmetric "bursty" traffic.

overhead. Moreover, the results in Fig. 6 show that the delay of MVM policy is smaller than that of the MWM policy. MWM-α policies have been studied in the literature [10], [16] and have been reported to incur smaller delay as the value of α goes to zero. Our simulations confirm this observation and also show that the delay performance of the MVM policy is no worse than the MWM-α policies even for small values of α.

Fig. 7 shows the delay for the bursty arrival process described above. The delay is significantly higher for the more bursty arrival process as compared to Bernoulli traffic. It seems that although the MVM and MWM-0+ policies have different tie-breaking rule, their delay performance is actually quite similar.

References

[1] A. Brzezinski, G. Zussman, and E. Modiano. Enabling distributed throughput maximization in wireless mesh networks: A partitioning approach. In *ACM Mobicom*, pages 26–37, New York, NY, USA, 2006. ACM Press.

[2] J. Dai and B. Prabhakar. The throughput of data switches with and without speedup. In *IEEE Infocom*, March 2000.

[3] J. G. Dai. On positive harris recurrence of multiclass queueing networks: a unified approach via fluid limit models. *Annals of Applied Probability*, 5:49–77, 1995.

[4] A. Dimakis and J. Walrand. Sufficient conditions for stability of longest-queue-first scheduling: Second-order properties using fluid limits. *Advances in Applied Probability*, 38(2):505–521, 2006.

[5] A. Feldmann, N. Kammenhuber, O. Maennel, B. Maggs, R. D. Prisco, and R. Sundaram. A methodology for estimating interdomain web traffic demand. In *IMC*, 2004.

[6] A. Hoffman and H. Kuhn. Systems of distinct representatives and linear programming. *The American Mathematical Monthly*, 63, 1956.

[7] C. Joo. A local greedy scheduling scheme with provable performance guarantee. In *ACM Mobihoc*, May 2008.

[8] C. Joo and N. B. Shroff. Performance of random access scheduling schemes in multi-hop wireless networks. In *IEEE Infocom*, May 2007.

[9] R. Karp and J. Hopcroft. An n/sup 5/2/ algorithm for maximum matchings in bipartite graphs. *SIAM Journal on Computing*, 2, 1973.

[10] I. Keslassy and N. McKeown. Analysis of scheduling algorithms that provide 100% throughput in input-queued switches. In *Proceedings of the 39th Annual Allerton Conference on Communication, Control, and Computing*, October 2001.

[11] I. Keslassy, R. Zhang-Shen, and N. McKeown. Maximum size matching is unstable for any packet switch. *IEEE Communications Letters*, 7, October 2003.

[12] J. M. Hall. Distinct representatives of subsets. *Bulletin of the American Mathematical Society*, 54, 1948.

[13] N. McKeown, V. Anantharam, and J. Walrand. Achieving 100% throughput in an input-queued switch. In *IEEE Infocom*, March 1996.

[14] A. Mekkittikul and N. McKeown. Practical scheduling algorithm to achieve 100% throughput in input-queued switches. In *IEEE Infocom*, April 1998.

[15] H. Perfect and J. Pym. An extension of banachs mapping theorem, with applications to problems concerning common representatives. *Proceedings of the Cambridge Philosophical Society (Mathematical and Physical Sciences)*, 62, 1966.

[16] D. Shah and D. Wischik. Optimal scheduling algorithms for input-queued switches. In *INFOCOM*, 2006.

[17] T. H. Spencer and E. W. Mayr. Node weighted matching. In *Proceedings of the 11th Colloquium on Automata, Languages and Programming*, pages 454–464, London, UK, 1984. Springer-Verlag.

[18] L. Tassiulas. Scheduling and performance limits of networks with constantly changing topology. *IEEE Trans. Inform. Theory*, 43:1067–1073, 1997.

[19] T. Weller and B. Hajek. Scheduling nonuniform traffic in a packet-switching system with small propagation delay. *IEEE/ACM Trans. Netw.*, 5(6):813–823, 1997.