# Degree-Sequenced Matching Algorithms for Input-Queued Switches

*Madhusudan Hosaagrahara* and *Harish Sethu*

Computer Communications Laboratory

Department of Electrical and Computer Engineering

Drexel University

3141 Chestnut Street

Philadelphia, PA 19104-2875.

Email: {madhu,sethu}@ece.drexel.edu

**Abstract**

This paper presents a class of algorithms for scheduling packets in input-queued switches. As opposed to previously known algorithms that focus only on achieving high throughput, these algorithms seek to achieve low average delay without compromising the throughput achieved.

Packet scheduling in input-queued switches based on the virtual-output-queued architecture is a bipartite graph matching problem wherein ports are represented by vertices and the traffic flows by the edges. The set of matched edges determine the packets that are to be transferred from the input ports to the output ports. Current matching algorithms implicitly prioritize high-degree vertices, i.e., ports with a large number of flows, causing longer delays at ports with a smaller number of flows. Motivated by this observation, we present three matching algorithms based on explicitly prioritizing low-degree vertices and the edges through them. Using both real gateway traffic traces as well as synthetically generated traffic, we present simulation results showing that this class of algorithms achieves a low average delay as compared to other scheduling algorithms of equivalent complexity while still achieving similar throughput. We also show that these algorithms determine the maximum size matching in almost all cases.

**Keywords**

Input-Queued Switches, Scheduling and Matching Algorithms

This work was supported in part by NSF CAREER Award CCR-9984161.

A preliminary version of this paper was presented at the 2005 IEEE Int'l Conference on Computer Communications and Networks, San Diego, USA.

## I. Introduction

A typical switch or a router in a network consists of a set of input ports that receive packets through incoming links, a set of output ports that transmit packets onto the outgoing links, a switching fabric (usually a crossbar) that transfers packets from the input ports to the output ports and some memory for buffering packets. Almost all the switch architectures that have been developed can be categorized based on where the incoming packets are buffered. The architecture that offers the highest throughput and low delay is an output-queued switch where packets arriving at the input ports are immediately transferred to their destination output port and, if necessary, buffered there. This requires packet buffers at the output ports to operate faster than the line rate since they must, in the worst case, buffer multiple packets arriving simultaneously from all the input ports (Karol et al., 1987). This problem is further compounded by the advent of high speed optical links with line rates several orders of magnitude higher than current memory speeds. This trend is expected to continue and place renewed emphasis on the design and development of switch and router architectures capable of handling terabits of data per second.

On the other hand, input-queued switches based on the virtual-output-queueing (VOQ) architecture do not require the buffers to operate faster than the line rates (Tamir and Frazier, 1988). Packets arriving at the input ports of a VOQ switch are buffered at the input ports in different queues based on their destination output port, thus requiring the packet buffers to operate only at the line rate. More recently, researchers have proposed using a switching fabric with buffered crossbars, i.e., with buffers at each cross-point, and demonstrated that this architecture is capable of achieving 100% throughput (Javidi et al., 2001). However, the buffered crossbar architecture does not scale very well because of current limitations on the number and size of memories that can be implemented on a single chip. The VOQ architecture, therefore, remains an appealing design for high-speed switches. Recent research has shown that it is possible to achieve 100% throughput as well as support Quality-of-Service (QoS) guarantees using input-queued switches (McKeown et al., 1999). Central to this success are the scheduling algorithms that control access to the switch fabric and determine which packets are transferred from the input ports to the output ports.

This paper proposes a new class of algorithms to schedule the transfer of packets from input ports to output ports in a VOQ switch. In contrast to past research which has largely focused on either high throughput or guaranteed services, this work seeks to achieve lower average delays for best-effort traffic while also achieving a throughput that is very close to the maximum possible.

## A. Background

Scheduling the transfer of packets from input ports to output ports in a VOQ switch is a bipartite graph matching problem. The input ports and the output ports form the two disjoint sets of vertices of the bipartite graph. The edge set of the graph is formed by adding an edge between two vertices if and only if the input port represented by one vertex has a packet queued for transmission through the output port represented by the other vertex. The scheduling problem is to generate a subset of edges such that each vertex has at most one edge incident on it. This subset of edges describes the set of packets that should be transferred from the input ports to the output ports.

It is desirable that the largest subset of edges be selected to maximize the overall throughput of the switch. The fastest known algorithm for Maximum Size Matching (MSM) in bipartite graphs was proposed by Hopcroft and Karp (1973). The algorithm begins with an initial match and then iteratively identifies an augmenting path; i.e., a simple path whose edges are alternately matched and unmatched and whose starting and ending vertices are unmatched. It increases the size of the initial matching by removing the matched edges that occur in the path from the initial matching and adding the unmatched edges of the path to the matching. This process is repeated until no augmenting path can be found. Although the algorithm guarantees the largest matching possible and therefore maximizes the overall throughput, it is unfair since it may lead to starvation of flows (i.e., some input and output ports are never matched). In addition, it has been shown to be unstable even for admissible traffic patterns; i.e., even if none of the input and output ports are over-subscribed, a queue may not become empty with probability one after some finite time (Keslassy et al., 2003). On the other hand, it has been proved that a maximum weight matching, with the weight function based either on the queue length (called the Longest Queue First or LQF algorithm) or on the queueing delay of the head-of-line packet (called the Oldest Cell First or OCF algorithm), is stable for all admissible traffic patterns (McKeown et al., 1999). It has also been shown that these algorithms can provide bandwidth and delay guarantees (Charny et al., 1998, Kam and Siu, 1999, Leonardi et al., 1999).

While the algorithms discussed above are stable and guarantee high-throughput, their implementation and time complexity make them unsuitable for implementation in high-speed switches. Therefore, the design of simpler algorithms that seek to approximate these perfect algorithms has received significant attention over the last decade. The need for simple, efficient and fast schedulers for high-speed switches was first addressed by Anderson et al. (1993) with the Parallel Iterative Matching (PIM) algorithm, a distributed maximal matching algorithm based on iterative negotiations between input and output ports.

The PIM scheduling algorithm is composed of multiple iterations, with each iteration having three distinct phases.

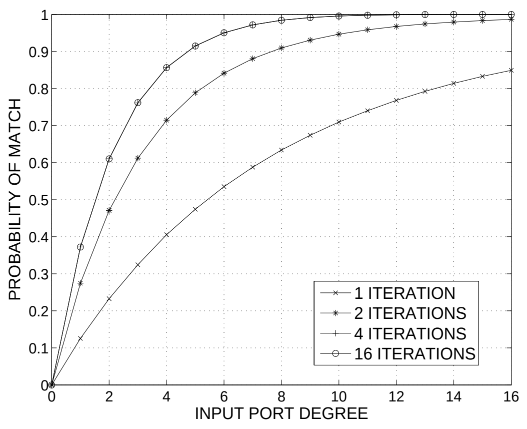
- The *Request* Phase: In this phase, the input ports first identify the output ports to which they have at least one packet queued and proceed to send a *Request* signal to these output ports asking for permission to transmit.
- The *Grant* Phase: The output ports receive these *Request* signals from the input ports. Each output port randomly selects one of the requesting input ports to which it sends a *Grant* signal indicating permission to transmit.
- The *Accept* Phase: The input ports receive these grants (an input port may receive more than one grant). Each input port randomly selects one of the grants and sends an *Accept* signal to the corresponding output port indicating acceptance of the grant.

The above three phases complete a single iteration. Ports between which an *Accept* signal was exchanged during an iteration are considered matched and they do not take part in subsequent iterations. PIM has been shown to achieve over $98\%$ utilization under uniform traffic in just $4$ iterations. However, since PIM is a randomized algorithm, it cannot place a finite upper bound on the length of time a flow might be starved.

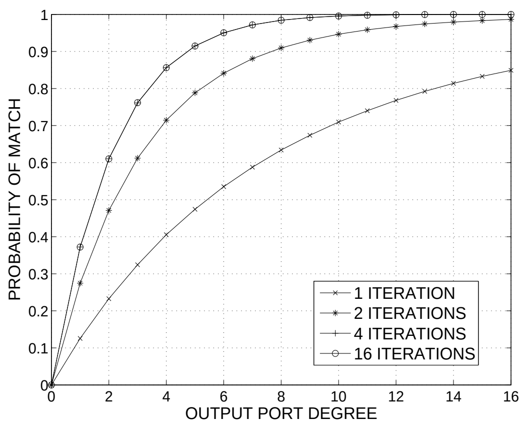
One approach to prevent starvation is to have the input and the output ports issue requests and grants in a round-robin fashion, with each output port issuing grants to the input that appears next in a fixed round-robin schedule. Similarly, the input ports issue accepts to the output port that appears first in a fixed round-robin schedule. While the round-robin matching algorithm is simple to implement and eliminates the use of randomness, it becomes unstable for an offered load of just $63\%$. This is because, at heavy loads, the counters move in lock-step and hence multiple output ports issue grants to a single input port.

The *i*SLIP algorithm, proposed by McKeown (1999), eliminates the use of randomness and solves the synchronization problem by having the input ports and the output ports issue grants and accepts in a round-robin fashion. Each port maintains a counter that keeps track of the next port to which a grant or an accept is to be issued. Input ports increment the counter to one past the last accepted output port, while output ports increment their counter by one past the input port it last granted to, if and only if the grant was accepted by the input port in the *Accept* phase. This round-robin fashion of issuing grants and accepts places a finite upper bound on the length of time a flow might be starved. McKeown (1999) also showed that *i*SLIP can provide $100\%$ throughput for uniform traffic.

Although the *i*SLIP algorithm bounds the starvation experienced by flows, it treats all flows equally and does not support bandwidth reservations. Zhang and Bhuyan (2003) proposed the Iterative Port-



(a) Probability of an input port being matched is directly proportional to its degree.



(b) Probability of an output port being matched is directly proportional to its degree.

Fig. 1. Simulation results illustrating PIM's implicit bias towards matching ports with a high degree.

based Deficit Round Robin (IPDRR) algorithm as a scheme to ensure that flows receive an allocation in proportion to their reservations. IPDRR maintains a Deficit Round Robin (DRR) scheduler at each input and output port (Shreedhar and Varghese, 1996). The reservation of a flow is determined by the quota of the flow in the DRR scheduler. An unmatched output port issues its grant to a requesting input port that occurs first in the linked list maintained by the DRR scheduler. Similarly, an unmatched input port issues its accept to the granting output port that occurs first in its linked list. The IPDRR algorithm also achieves $100\%$ throughput under uniform traffic.

## *B. Motivation and Contributions*

Most matching algorithms proposed for scheduling packets in an input-queued switch have focused on achieving $100\%$ throughput or at least as high a throughput as possible. When throughput is the primary goal, it is often the case that some ports will more likely to be matched than others. For example, in the PIM scheduling algorithm, an input port with a high degree will issue a greater number of requests and have a greater probability of receiving a grant, as compared to an input port of lower degree. Similarly, an output port with high degree is more likely to receive a request from an input port of smaller degree, and hence has a greater probability of receiving an accept. Fig. 1 plots the fraction of instances in which a port of a certain degree was matched based on simulations of the PIM scheduling algorithm on $2,500,000$ randomly generated bipartite graphs. Fig. 1(a) shows that an input port with a large degree has a significantly higher probability of being matched when compared to a port of a smaller degree. Similarly, as Fig. 1(b) shows, PIM is more likely to match a high-degree output port. This observation holds true even if the number of iterations performed are increased; when $16$ iterations are performed, a port with a degree of $8$ is almost guaranteed to be matched, whereas a port with a degree of $1$ has only a $40\%$ probability of being matched.

Further, as we shall show later in this paper, high throughput does not necessarily imply low average delays. Mechanisms that achieve guaranteed services such as a delay deadline also do not necessarily achieve low average delays. Meanwhile, low average delays combined with high-throughput is the ideal goal for best-effort traffic.

This paper seeks to achieve low average delays while also achieving very high throughput in VOQ switches. Our approach is motivated by the fact that randomized algorithms frequently bias the matching algorithm in favor of high-degree ports, causing higher delays for packets at ports corresponding to low-degree ports. Matching the high-degree ports first reduces the number of edges available to vertices with a smaller degree and hence, reduces the size of the matching.

Borrowing from a strategy used by Angluin and Valiant (1977), we describe three algorithms for scheduling packets in a VOQ switch. These algorithms are based on the principle of matching ports with the lowest degree first, thus increasing the choice available in later iterations and resulting in large matchings. The first algorithm, called *Degree-Sequenced Matching*, sorts vertices in order of their degrees and, in each iteration, matches the vertex with the lowest degree to one of its neighbors. We show that this approach leads to lower average delays without compromising the throughput goals. The algorithm requires no state information to be carried over for the next matching since the degree information is already available at the input and the output ports. We investigate the scenarios in which DSM calculates a match smaller than the maximum possible match and present two improved versions of the algorithm. The first version, called the *enhanced Degree-Sequenced Matching*, identifies pairs of vertices with the lowest degree and matches them first. The second version, called the *neighbor-aware Degree-Sequenced Matching*, improves upon the earlier versions by considering both the degrees of the vertices as well as the set of neighbors of the vertices. We examine, via simulations, the trade offs between the performance and complexity that these algorithms provide.

A similar approach was used by Gura and Eberle (2002) in developing the Least Choice First (LCF) scheduling method. The LCF algorithm follows the approach of the *i*SLIP algorithm in issuing grants and accepts. If, however, an output port does not receive a request from the input port its counter points to, then the output port issues a grant to the input port with the smallest degree amongst the requests it has received. Similarly, an input port first accepts a grant from the output port its counter points to. If the input port does not receive a grant from that output port, then it accepts the grant issued by the output port with the smallest degree. While the LCF algorithm improves upon the performance of PIM and *i*SLIP scheduling algorithms, it reduces to the *i*SLIP algorithm at high loads since each output port receives a request from all the input ports.

In our simulation studies, we plot the fraction of instances in which the DSM algorithm does not achieve the perfect matching for various sizes of switches and show that these fractions are very small with almost negligible impact on overall throughput. Using a detailed simulation environment, we show that the proposed algorithms achieve throughput close to or better than that achieved by most other scheduling algorithms. Further, we show that these algorithms achieve significantly better average delays than other algorithms of equivalent complexity. We present these results for both real gateway traffic traces as well as synthetically generated random traffic.

Section II presents our notation and the DSM algorithm in detail along with its pseudo-code description. The two improvements to DSM are presented in Sections III and IV. Simulation results are presented in Section V. Section VI concludes the paper with some closing remarks on the possible future directions.

## II. Degree-Sequenced Matching (DSM)

### *A. Notation*

We denote a bipartite graph by $G(U, V, E)$ where $U$ and $V$ are the disjoint sets of vertices and $E$ is the set of edges. Each edge $e \in E$ connects a vertex in $U$ to a vertex in $V$. The neighbors of a vertex $u$, which we denote by $\Gamma(u)$, are the set of vertices that $u$ is connected to. The degree $deg(u)$ of a vertex $u$ is the number of vertices it is connected to. Further, we denote the smallest degree of all the vertices in the graph by $\delta(G)$; i.e., $\delta(G) = \min(\{deg(u) : \forall u \in G\})$.

A matching $M$ on the graph $G$ is a subset of $E$ such that each vertex has at most one edge in $M$ incident on it. The matching problem is to identify $M$ and the maximum size matching problem is to identify the largest possible $M$. In the following, we assume that the vertex sets are of the same size, i.e., $|U| = |V| = n$ and that the graph contains $|E| = m$ edges.

### *B. Description*

The algorithm is iterative and in each iteration, it identifies the minimum degree vertex and matches it to one of its randomly selected neighbors. If there are multiple minimum degree vertices, one of them is selected at random. These two matched vertices are now eliminated from further processing and consequently, the degrees of each of the neighbors of these vertices is decreased by 1. In this process, any vertices that are disconnected from the rest of the graph are also eliminated. The iterations are repeated until there are no more pairs of vertices in the graph available to match.

```
01 **Initialize:**
02 create set $\mathcal{M} = \{\}$
03 create heap $\mathcal{H}$
04 *for each* vertex $v \in G$
05    *if* $deg(v) > 0$
06       insert tuple $\{(v, deg(v))\}$ into $\mathcal{H}$
07    *end if*
08 *end for*
09 **Match:**
10 *while* $\mathcal{H}$ is not empty
11    $\{(v, deg(v))\} \leftarrow$ minimum degree vertex in $\mathcal{H}$
12    randomly select a vertex $u$ such that $u \in \Gamma(v) \cap \mathcal{H}$
13    $\mathcal{M} \leftarrow \mathcal{M} \cup \{(u, v)\}$
14    remove $u$ and $v$ from $\mathcal{H}$
15    *for each* vertex $w \in \Gamma(u) \cup \Gamma(v)$
16       decrease key of $w$ in $\mathcal{H}$ by $1$
17       *if* $deg(w)$ is $0$
18          remove $w$ from $\mathcal{H}$
19       *end if*
20    *end for*
21 *end while*
22 *return* $\mathcal{M}$
```

Fig. 2. Pseudo-code for the centralized version of the Degree-Sequenced Matching (DSM) algorithm.

Fig. 2 presents the pseudo-code of the algorithm that uses a heap to identify the minimum degree vertices. Lines 01–08 describe the initialization phase wherein the heap is created and each vertex of the bipartite graph is inserted into the heap as a tuple along with its degree. In each iteration, the algorithm identifies the vertex with the smallest degree that is as yet unmatched (line 11) and matches it randomly with one of its neighbors (lines 12–13). The matched pair is now eliminated from the heap (line 14). Lines 15–20 update the degrees of all the neighbors of the two matched and eliminated vertices. Note that the degree of any neighbor is reduced by 1, since no vertex can be connected to both the matched vertices. Further, if any of the neighbors are no longer connected to the graph, i.e., if their degree is zero, then they are deleted from the heap. In such cases, we do not need to update the degrees of the neighbors of the deleted vertex since, if the degree of the deleted vertex was 1, then it was connected either to $u$ or to $v$. If its degree was greater than 1, then decreasing its degree by 1 does require updating the degrees of its neighbors.

### *C. Correctness*

We prove that the algorithm achieves a maximal match by first proving a loop-invariant that $\mathcal{M}$ and $\mathcal{H}$ are disjoint; i.e., a vertex is either a part of the matched set $\mathcal{M}$, or is a part of the unmatched heap $\mathcal{H}$ or neither. Initially, $\mathcal{M} = \{\}$ and $\mathcal{H} = V \cup U$. In each iteration, a vertex that is removed from $\mathcal{H}$ is either matched and inserted into $\mathcal{M}$ or is discarded because, after its degree was decreased to $0$, it was disconnected from the graph. At the end of the execution of the algorithm, $\mathcal{H} = \{\}$, thus proving the loop-invariant. Note that once a vertex is matched and removed from the heap, it is never revisited by the algorithm and all the other unmatched edges incident on it are removed from the graph, thus proving the correctness of the algorithm. Further, the algorithm returns a maximal match. This is because once two vertices are matched, they remain matched even if a better match is possible.

### *D. Performance*

Since the algorithm is a maximal matching algorithm, it is $\frac{1}{2}$-optimal. We, however, present a more formal proof.

Consider the operation of DSM on a bipartite graph that has a maximum size matching of size $S$. In each iteration, DSM selects two vertices and deletes all the edges incident on these vertices. At most two of these deleted edges could be present in $S$. In lieu, DSM adds the edge between these vertices to its matching. Thus, the matching produced by DSM must contain at least $\frac{S}{2}$ edges and DSM is, therefore, $\frac{1}{2}$-optimal.

Our simulation results with real traffic, however, indicate that the DSM algorithm achieves almost as good throughput as a maximum matching algorithm and better throughput than most other algorithms.

### *E. Complexity*

We derive the worst-case time-complexity assuming that the algorithm uses a Fibonacci Heap. The *Initialize* routine is of $O(n)$ complexity since it inserts at most $2n$ elements one by one into a heap, and the amortized cost of an insertion to a Fibonacci Heap is $O(1)$. The *while* loop executes at most $n$ times since in each iteration two elements are removed from the heap. The complexity of lines $11$ and $13$ is at most $\log n$ and the complexity of lines $12$ and $14$ is $O(1)$, and they are executed at most $n$ times. Line $16$ is executed at most $m$ times during the execution of the algorithm. Thus, the overall complexity of the algorithm is $O(m + n \log n)$. Note that $m$ is $O(n^2)$ but is much smaller in sparse graphs.

```
**Initialize:**
create set $\mathcal{M} = \{\}$
create heap $\mathcal{H}$
*for each* vertex $v \in G$
    *if* $deg(v) > 0$
        Create heap $h_v$
        *for each* vertex $u \in \Gamma(v)$
            Insert tuple $\{(u, deg(u))\}$ into $h_v$
        *end for*
        Insert tuple $\{(v, deg(v), h_v)\}$ into $\mathcal{H}$
    *end if*
*end for*
**Begin:**
*while* $\mathcal{H}$ is not empty
    $\{(v, deg(v), h_v)\} \leftarrow$ minimum degree vertex in $\mathcal{H}$
    $\{(u, deg(u))\} \leftarrow$ minimum degree vertex in $h_v$
    $\mathcal{M} \leftarrow \mathcal{M} \cup \{(u, v)\}$
    remove $u$ and $v$ from $\mathcal{H}$
    *for each* vertex $w \in \Gamma(u) \cup \Gamma(v)$
        *if* $u \in h_w$
            remove $u$ from $h_w$
        *else*
            remove $v$ from $h_w$
        *end if*
        decrease key of $w$ in $\mathcal{H}$ by $1$
        *if* $deg(w)$ is 0
            remove $w$ from $\mathcal{H}$
        *end if*
        *for each* vertex $x \in \Gamma(w)$
            decrease key of $w$ in $h_x$ by $1$
        *end for*
    *end for*
*end while*
```

Fig. 3. Centralized version of the enhanced Degree-Sequenced Matching Algorithm.

## III. ENHANCED DEGREE-SEQUENCED MATCHING (EDSM)

We now present an improvement to the Degree-Sequenced Matching algorithm. Recall that in each iteration DSM selects the minimum degree vertex and matches it to one of its neighbors. The neighbor is selected randomly and hence, when a very low-degree vertex is connected to a high-degree vertex, DSM may match a low-degree vertex with a high-degree vertex. Viewed from another perspective, a high-degree vertex has a greater probability of being connected to a low-degree vertex and hence may be matched early on, reducing the choice available during later iterations.

## *A. Description*

The algorithm presented in this section improves upon DSM by matching vertices to their neighbors with the lowest degree. In each iteration, it selects the minimum degree vertex and matches it to its minimum degree neighbor. If in any of the previous steps, multiple vertices have the smallest degree, one of them is randomly selected. The matched vertices are eliminated from further processing and the degrees of all their neighbors are reduced by $1$.

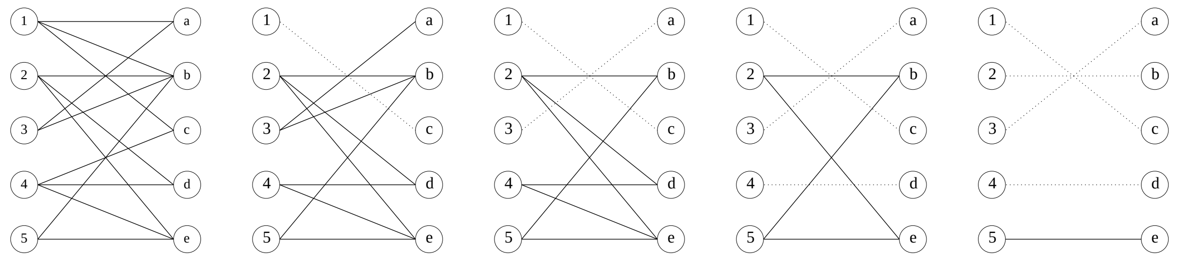
Fig. 3 presents the pseudo-code for this improved version. Since each vertex has to keep track of the degrees of its neighbors, the algorithm creates a min-heap for each vertex (line $06$) and all the neighbors of the vertex are inserted into this heap with a lower degree implying greater priority (lines $07$–$9$). All the vertices are then inserted into a heap (line $10$). In each iteration, the algorithm extracts the smallest degree vertex (line $15$). It also extracts the neighbor of that vertex with the smallest degree (line $16$). These two vertices are matched (line $17$) and hence eliminated from consideration in succeeding iterations (line $18$). Lines $19$–$24$ remove these matched vertices from all the heaps they occur in. Lines $25$–$32$ decrease the degrees of the neighbors of these matched vertices in all the heaps that they are present in. As in DSM, it is possible that reducing the degree of a vertex may disconnect it from the graph. For example, if $\{u, v, w\}$ and $\{(u, v), (w, v)\}$ are the vertices and edges, respectively, in one instance of the bipartite graph, then the algorithm will match $u$ (or $w$) to $v$ thus disconnecting the other vertex $w$ (or $u$) from the rest of the graph. Lines $26$–$28$ check for such a case and remove the disconnected vertex from the graph. This check is unnecessary in lines $29$–$31$, since if a vertex had a degree of $1$ before it was disconnected, then it would not have had any neighbors occur other than the matched vertex, and hence would not occur in any other heap.

## *B. Correctness*

The proof for correctness is similar to that of DSM and relies on the loop-invariant that $\mathcal{M}$ and $\mathcal{H}$ are disjoint. Initially, $\mathcal{M} = \{\}$ and $\mathcal{H} = V \cup U$. In each iteration, a vertex that is removed from $\mathcal{H}$ is either matched and inserted into $\mathcal{M}$ or is discarded because its degree was decreased to $0$. At the end of the algorithm, $\mathcal{H} = \{\}$. Further, a vertex that is matched is removed from all the heaps it occurs in. Hence, the algorithm never revisits a vertex that it previously matched.

## *C. Performance*

The algorithm is maximal and is $\frac{1}{2}$ optimal. The proof is similar to that of DSM.



(a) The initial traffic pattern in a 5-input 5-output port switch.

(b) The resulting traffic pattern after the first iteration where vertices 1 and c were matched.

(c) The resulting traffic pattern after the second iteration where vertices 3 and a were matched.

(d) The resulting traffic pattern after the third iteration where vertices 4 and d were matched.

(e) The resulting traffic pattern after the fourth iteration where vertices 2 and b were matched.

Fig. 4. An operation of the DSM algorithm that results in a maximum match.

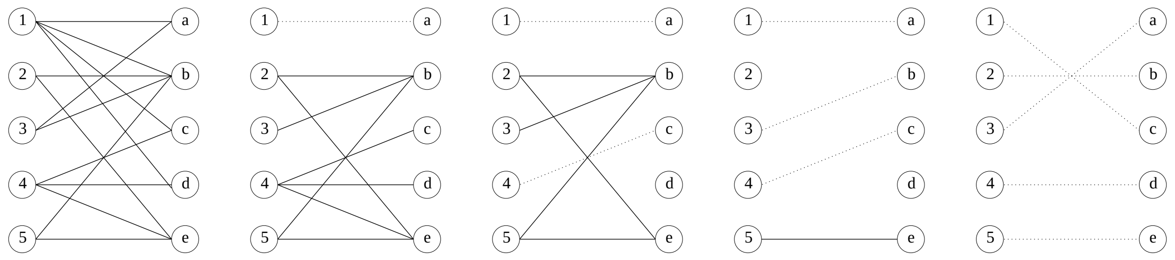
## *D. Complexity*

We assume that all the heaps are Fibonacci heaps. The initialization phase of the algorithm creates one main heap ($\mathcal{H}$) and a heap for each vertex. Thus, a total of $2n + 1$ heaps are created. Inserting all the vertices into the main heap requires $2n$ insertions. Further, the neighbors of each vertex are inserted into the heap associated with that vertex. The number of insertions required to insert the neighbors of each vertex into the heap associated with that vertex is twice the number of edges in the graph and equals $2m$ (or each edge requires 2 insertions). Hence, the total number of insertions in the initialization phase is $2n + m$ which is $O(m)$.

In each iteration, two vertices are removed from the main heap. Hence, the maximum number of iterations required are $n$ and these deletions total $2n$. Further, in each iteration, the matched vertices are removed from the heaps associated with each vertex. Initially, the total number of elements in all the heaps associated with the vertices is $2m$ and all these heaps are empty at the end of the algorithm. Hence, the total number of deletions are $O(m)$.

The total number of decrease key operations performed in each iteration is one less than the sum of the degrees of the matched vertices. One of these vertices is guaranteed to have the minimum degree, while the other vertex will have a degree greater than or equal to the minimum degree and less than or equal to the maximum degree of the graph. Thus, in the worst case, the total number of decrease key operations is $O(m)$.

The total time complexity of the algorithm is thus $O(m) + O(m \log n) + O(m)$ which is $O(m \log n)$.



(a) The initial traffic pattern for a 5-input 5-output port switch.
(b) The resulting traffic pattern after vertices *1* and *a* were matched in the first iteration.
(c) The resulting traffic pattern after vertices *4* and *c* were matched in the second iteration.
(d) The resulting traffic pattern after vertices *3* and *b* were matched in the third iteration.
(e) One possible matching of size 5 for the traffic patter.

Fig. 5. A sample traffic pattern for which DSM does not return a maximum match.

## IV. neighbor-aware Degree-Sequenced Matching(nDSM)

To explain the motivation for this third algorithm, we shall examine the reasons why DSM and eDSM may not determine the maximum match. Consider the operation of DSM on the traffic pattern depicted in Fig. 4(a). In the first iteration, vertices *3*, *5*, *a*, *c*, and *d* have the smallest degree. One of these, say vertex *c*, is randomly selected and is matched to its one of its neighbors, say vertex *1*, resulting in the residual traffic pattern depicted in Fig. 4(b). At this point, vertex *a* has the smallest degree; it is selected and matched to vertex *3*, resulting in the residual traffic pattern depicted in Fig. 4(c). Vertices *4* and *2* are matched to vertices *d* and *b* in the third and fourth iterations, respectively, leaving vertices *5* and *e* to be matched in the fifth and final iteration.

Now consider the operation of DSM on the traffic pattern depicted in Fig. 5(a). At the beginning of the first iteration, vertices *3*, *5*, *a*, *c*, and *d* have the smallest degree. Assume that vertex *a* is selected and matched to vertex *1* in the first iteration. Thus, at the end of the first iteration, vertices *c* and *d* have a degree of 1 and are both connected to vertex *4*. Successive iterations match vertices *4* and *c*, *3* and *b*, and *5* and *e* resulting in a matching of size 4. The maximum possible match, however, is of size 5 as depicted in Fig. 5(e).

Thus, DSM may potentially return sub-optimal matches for some traffic patterns. The key to understanding this phenomenon is in the traffic pattern itself, wherein vertices *c* and *d* both have the smallest degree and have the same set of neighbors. Hence, in the first iteration when vertices *1* and *a* are matched, vertices *c* and *d* are connected to the same vertex and succeeding iterations can match only one of these two vertices, thus resulting in a smaller match. In fact, in our study we found that in most of the cases in which DSM or eDSM did not result in a maximum match, the traffic pattern contained a set of vertices

```
Initialize:
create set $\mathcal{M} = \{ \}$
Match:
while G is not empty
    $S_u = \{u : u \in U \texttt{ and } deg(u) = \delta(G)\}$
    $S_v = \{v : v \in V \texttt{ and } deg(v) = \delta(G)\}$
    $S = S_v \cup S_u$
    $Q_u = \{u : u \in U_s \texttt{ and } \forall\, u' \in Q_u, \Gamma(u) = \Gamma(u')\}$
    $Q_v = \{v : v \in V_s \texttt{ and } \forall\, v' \in Q_v, \Gamma(v) = \Gamma(v')\}$
    $Q = Q_u \cup Q_v$
    if Q is empty
        $u \leftarrow$ select a vertex from $S$
        $v \leftarrow$ minimum degree vertex in $\Gamma(u)$
    else
        $u \leftarrow$ select a vertex from $Q$
        $v \leftarrow$ minimum degree vertex in $\Gamma(u)$
    end if
    $\mathcal{M} \leftarrow \mathcal{M} \cup \{(u, v)\}$
    $G \leftarrow G \backslash \{u, v\}$
    end if
return $\mathcal{M}$
```

Fig. 6. Centralized version of the neighbor-aware Degree-Sequenced Matching (nDSM) algorithm.

that had the smallest degree and shared the same set of neighbors. This insight forms the basis of the third algorithm.

## A. Notation

Let $S_u$ and $S_v$ denote the set of vertices in $U$ and $V$, respectively, whose degree equals $\delta(G)$. Formally, $S_u = \{u : u \in U \texttt{ and } deg(u) = \delta(G)\}$ and $S_v = \{v : v \in V \texttt{ and } deg(v) = \delta(G)\}$. $S_u$ and $S_v$ shall together be referred to as $S = S_u \cup S_v$.

Let $Q_u$ and $Q_v$ denote a set of vertices in $S_u$ and $S_v$, respectively, that share the same set of neighbors; i.e., $Q_u = \{u : u \in S_u$ and $\forall u' \in Q_u, \Gamma(u) = \Gamma(u')\}$ and $Q_v = \{v : v \in S_v$ and $\forall v' \in Q_v, \Gamma(v) = \Gamma(v')\}$. $Q_u$ and $Q_v$ shall together be referred to as $Q = Q_u \cup Q_v$.

Note that for any graph the set $S$ is unique. However, the set $Q$ need not be unique since different subsets of $S_u$ and $S_v$ may qualify as instances of $Q_u$ or $Q_v$, respectively. In such cases, it is sufficient to arbitrarily select $Q_u$ and $Q_v$ from the possible candidates and generate the set $Q$.

### B. Description

This version of DSM, which we call *neighbor-aware DSM* (nDSM), checks if any of the vertices with the smallest degree share the same set of neighbors. If they do, then nDSM matches these vertices with one of the neighbors; otherwise, if no such vertices exist, nDSM reduces to eDSM and matches a vertex with the smallest degree to its smallest degree neighbor.

Fig. 6 presents the pseudo-code for nDSM. In each iteration, the algorithm first determines the vertices with the smallest degree (lines 05–07). It then identifies any vertices that, in addition to having the smallest degree, share the set of neighbors (lines 08–12). If multiple sets of such vertices exist, nDSM selects one such set. If no such vertices exist, then nDSM reduces to eDSM. It selects a vertex from the set of vertices of the smallest degree and matches it to its smallest degree neighbor (lines 11–13). If, on the other hand, some vertices share the same set of neighbors, then nDSM matches one of these vertices to its smallest degree neighbor (lines 15–17). nDSM then updates the graph by eliminating these matched vertices, all the edges incident on these vertices, and any vertices that are disconnected from the rest of the graph during this process of elimination; if there are vertices left to match, nDSM continues onto the next iteration.

### C. Correctness

We shall prove correctness by showing that $G$ and $M$ are disjoint. Initially, all the vertices are unmatched and are in $G$. $M$ is empty. In each iteration, the algorithm removes two vertices from $G$ and inserts them into $M$. Hence, a vertex that is matched is never revisited. Further, lines 13 and 16 ensure that the vertices are always disjoint. Hence the algorithm returns a matching.

### D. Performance

The algorithm is maximal and hence, is $\frac{1}{2}$ optimal. The formal proof is similar to that of DSM.

### E. Complexity

The minimum degree of the graph, $\delta(G)$, and the sets, $S_u$ and $S_v$ can be determined in $O(m)$ time. In the worst case, determining $Q_u$ and $Q_v$ will take $n^2$ time. The algorithm requires $n$ iterations, leading to a total complexity of $O(n^3)$.

The above bound is weak. First, nDSM does not need to determine all the vertices that could be in $Q$; in fact, it is sufficient to identify only two minimum degree vertices that share the same set of neighbors. In the worst case, however, it will need to compare the neighbors of all the vertices to ensure that $Q$ is empty.

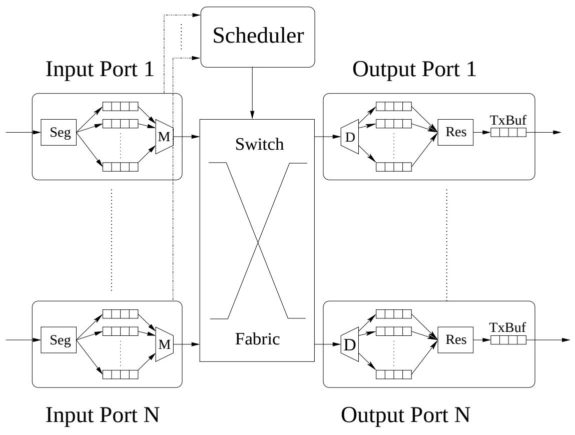


Fig. 7. Architecture of the input-queued switch used for simulation.

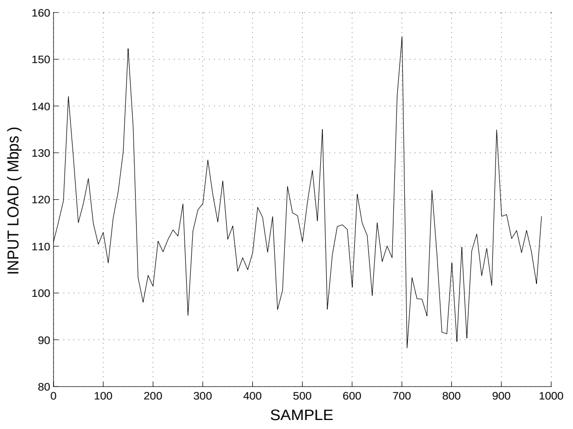
A hardware implementation of nDSM can perform this comparison faster by having each vertex maintain an $n$ bit vector, wherein a bit is set if that vertex has an edge to the corresponding vertex and unset otherwise. $Q$ can be determined by testing if any two of these vectors are equal. This test can be performed in $O(1)$ time using $O(n^2)$ comparators. Similarly, the minimum of the degrees of the ports can be determined in $O(\log n)$ time using $O(n \log n)$ comparators.

## V. Simulation Results

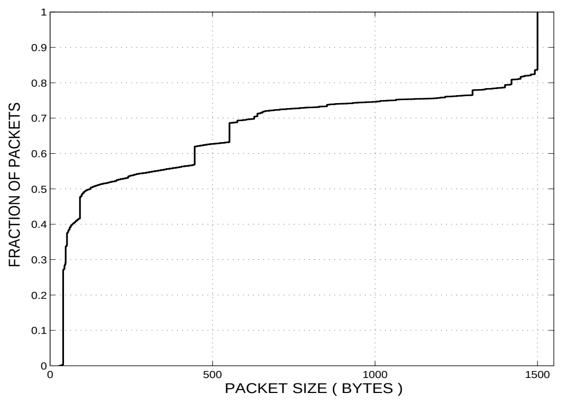
Our simulation experiments assume an $n$-input, $n$-output input-queued switch that uses virtual output queueing, as shown in Fig. 7. IP packets arriving at each input port are first broken down into fixed-size cells of $48$ bytes by the segmentation (*Seg*) module, after which they are placed into the virtual output queue (VOQ) for their destination port.

Each input port keeps the *Scheduler* updated of the state of its virtual output queues and the *Scheduler* is responsible for configuring the crossbar for each transmission slot. Cells arriving at an output port are first stored in the re-assembly buffers. When the last cell of a packet arrives into the re-assembly buffers, the re-assembly module (*Res*) extracts all the cells, reassembles them into the complete IP packet and enqueues the packet into the transmission queue (*TxBuf*). The transmitter, which is not shown in this figure, dequeues packets from this queue and sends them out on the link. *M* and *D* are multiplexers and demultiplexers, respectively, which control access to the queues. Sufficient control logic is assumed to be present at the ports to manage the multiplexers and demultiplexers and related control signals.

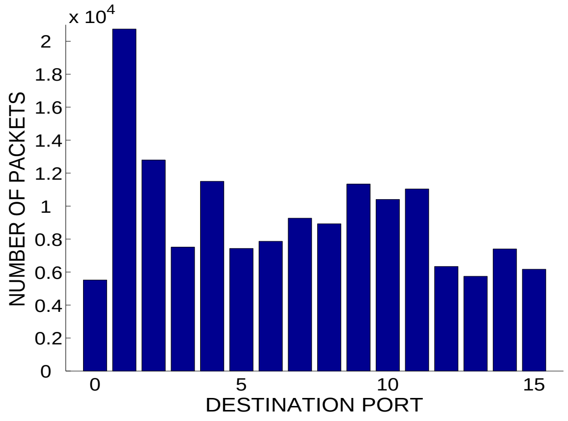
We report simulation results for a 16-port switch operating with a per-port bandwidth of $155$ Mbps using nine different scheduling algorithms listed below:



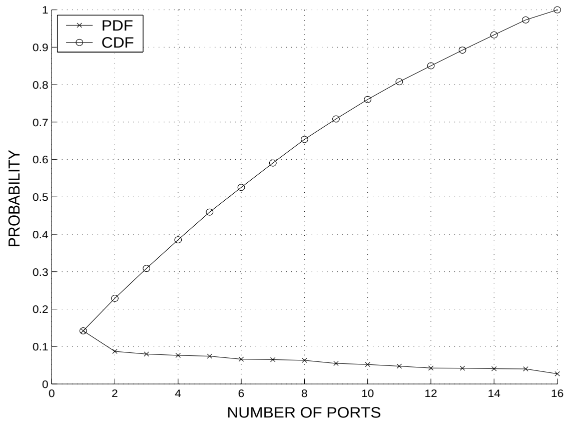
(a) Instantaneous input load of the real traffic traces averaged over 100 clock ticks.



(b) Cumulative Distribution Function (the fraction of packets smaller than any given size) of the packet sizes in the real traffic traces.



(c) Representative destination port distribution.



(d) Representative source port distribution.

Fig. 8. Characteristics of the real traffic traces from NLANR.

- PIM, for its historical significance
- *i*SLIP, also for its historical significance
- IPDRR, for its ability to provide guaranteed QoS and its superior performance over the iterative Fair Scheduler (*i*FS) (Ni and Bhuyan, 2002, Zhang and Bhuyan, 2003). Since all the flows were best-effort traffic, each flow is assigned a weight of 1.
- The maximum size matching (MSM) algorithm, for its promise of maximum throughput.
- A variant of maximum weight matching called the Longest Queue First (LQF), where the edge weights are equal to the number of cells in the corresponding VOQ, for its stability (McKeown et al., 1999).
- Another variant of maximum weight matching called the Oldest Cell First (OCF) algorithm, where

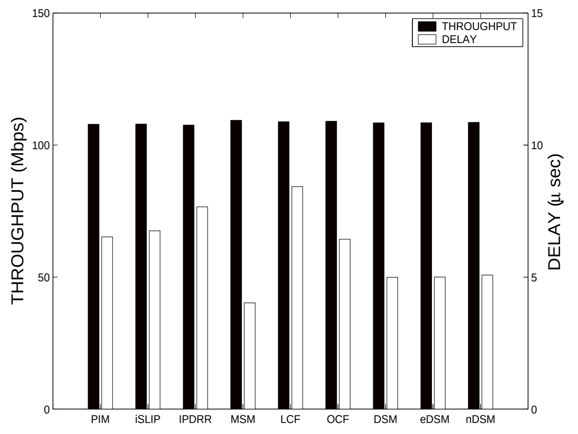


Fig. 9. Throughput and average delay with real traffic traces obtained from NLANR.

the edge weights are equal to the queueing delay of the head-of-line packet in the corresponding VOQ, also for its stability (McKeown et al., 1999).

- Finally, the algorithms presented in this paper, i.e., Degree-Sequenced Matching (DSM), eDSM and nDSM.

To ensure maximum possible performance, iterative algorithms, namely PIM, *i*SLIP, and IPDRR, were run for 16 iterations.

We present simulation results using both real and artificial traffic traces. Real traffic traces are available at the web-site of the Passive Measurement and Analysis (PMA) project of the National Laboratory for Applied Network Research (NLANR). The traces are from the NLANR Colorado State University *OC3c* tap (2003) that uses a pair of network cards synchronized by cable to tap an *OC3c* link. The original trace file contains traces of about $2.5 \times 10^6$ packets, representing about 90 seconds of network activity. In order to ensure that the average traffic rate arriving at each port was approximately equal, 16 traces were derived by time-slicing the original trace file into 16 separate traces, each containing about $1.5 \times 10^5$ packets.

Figs. 8(a) and 8(b) describe some characteristics of the real traffic traces obtained from NLANR. The average input traffic rate is around 112 Mbps per port although the instantaneous load varies significantly from 90 Mbps to 155 Mbps. Fig. 8(a) plots the average input load observed during each interval of 100 clock ticks. The size of the packets varies from 64 bytes to 1500 bytes with a significant fraction of packets being either 64 or 1500 bytes long. Fig. 8(b) shows the cumulative distribution function of the sizes of packets. The destination output port of each packet is determined by the destination IP address of

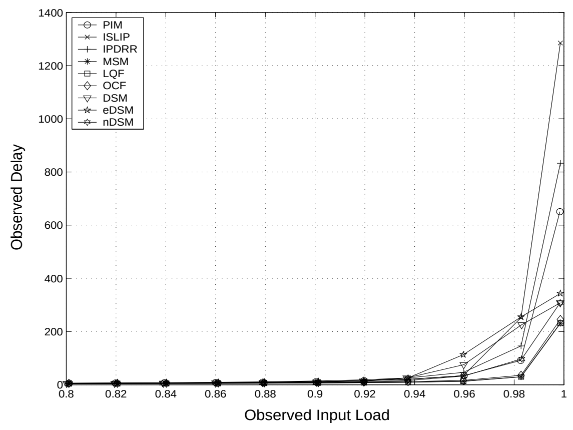


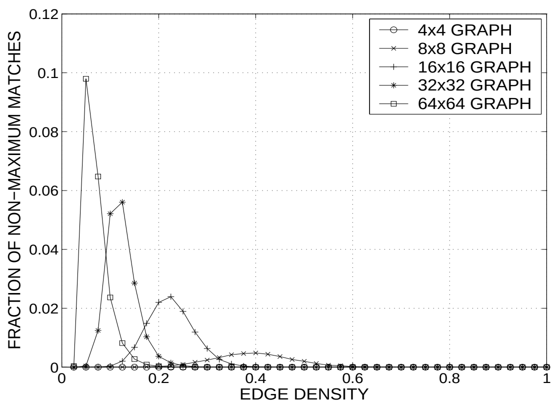
Fig. 10. Input load (as a fraction of the line rate) versus average delay in the region of interest using artificially generated uniform Poisson traffic.

the packet. The traffic is non-uniform, with different output ports receiving different percentages (ranging from $5\%$ to $15\%$) of the total traffic arriving into any given input port. Fig. 8(c) shows a representative destination distribution of packets arriving at an input port. This non-uniformity of the traffic is also evident in Fig. 8(d) which plots the distribution of the sources of packets arriving at a representative output port. Almost $15\%$ of the packets arriving at an output port originate from a single input port.

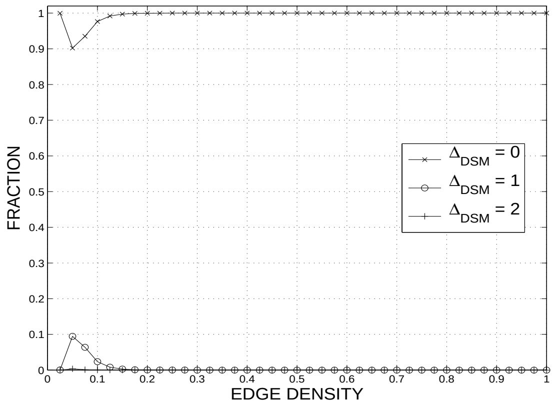
In order to analyze the performance characteristics, we also present results of simulations using synthetic traffic. Poisson traffic at fractions varying from $0.1$ to $1.0$ of the line rate were generated using a custom traffic generator with the packet destinations being uniformly distributed. Packet sizes were assumed to be a constant equal to $48$ bytes. Hence, simulations performed using synthetic traffic assume that the switch operates with an internal per port bandwidth of $48 \times 8 = 384$ Mbps with no speedup.

Fig. 9 shows the throughput and the average delay under real gateway traffic of the three algorithms as compared to other algorithms. Note that the performance of these algorithms is close to that of the maximum size matching (MSM), theoretically the best performing algorithm but with a complexity of $O(n^{\frac{5}{2}})$. Our DSM algorithm, however, has a complexity of only $O(n^2)$ for dense graphs and significantly lower complexity for sparse graphs. The figure also shows that the throughput achieved by the various algorithms under real traffic is almost identical. However, a significant variation is seen in the average delays achieved. Our algorithms achieve a lower average delay than all other algorithms except MSM.

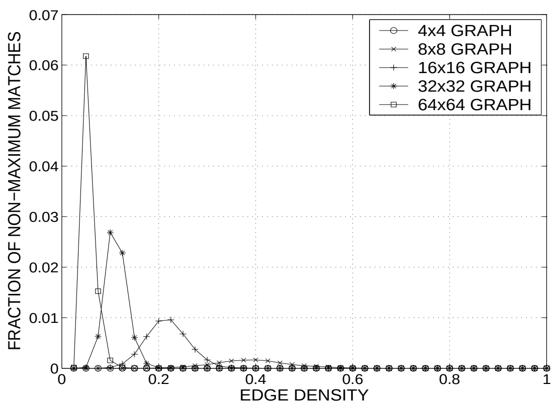
Fig. 10 plots the delay versus the input load for all the schedulers under synthetically generated Poisson traffic. Packet destinations were uniformly distributed amongst all the output ports, thus minimizing



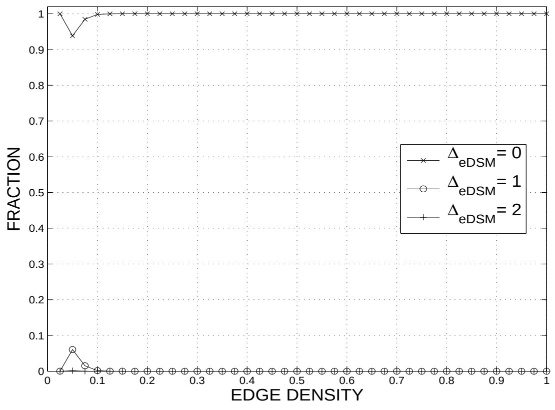
(a) Fraction of graph instances which do not yield a maximum match with DSM.



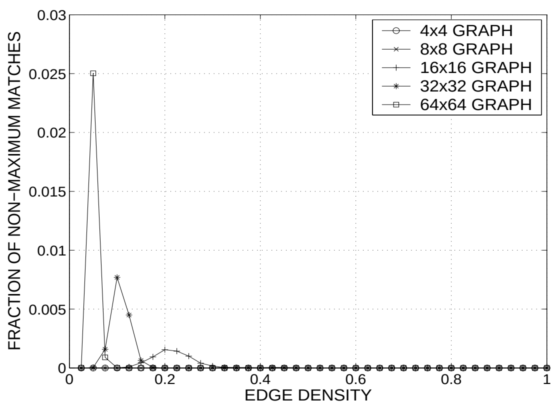
(b) Fraction of instances of a $64 \times 64$ graph in which DSM yields a match that is exactly $\Delta_{\text{DSM}}$ less than the maximum size match.



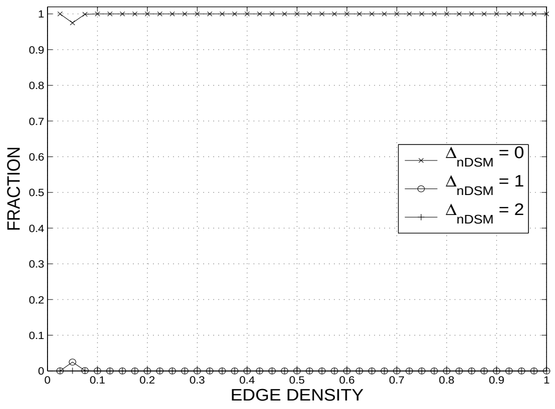
(c) Fraction of graph instances which do not yield a maximum match with eDSM.



(d) Fraction of instances of a $64 \times 64$ graph in which eDSM yields a match that is exactly $\Delta_{\text{eDSM}}$ less than the maximum size match.



(e) Fraction of graph instances which do not yield a maximum match with nDSM.



(f) Fraction of instances of a $64 \times 64$ graph in which nDSM yields a match that is exactly $\Delta_{\text{nDSM}}$ less than the maximum size match.

Fig. 11. Comparison of the three algorithms with the maximum size matching algorithm.

contention. Further, in order to achieve the maximum possible performance, PIM, *i*SLIP and IPDRR were run with $16$ iterations. All the algorithms perform well under these benign conditions. However, when the traffic load approaches unity our algorithms perform better than PIM, *i*SLIP and IPDRR and only negligibly worse than MSM, OCF and LQF.

It is of interest to see in what instances and how often our algorithms result in a match that is less than the maximum possible. We study the performance of these algorithms on randomly generated bipartite graphs of different sizes ($4\times4$ to $64\times64$) and edge densities (fraction of vertex pairs with an edge between them). The fraction of instances in which DSM, eDSM, and nDSM fail to return a perfect match are shown in Figs. $11$(a), $11$(c), and $11$(e), respectively. It may be noted that the algorithms return a maximum match for almost all edge densities. For example, in a $64\times64$ graph, DSM always returns the maximum matching when the edge density is greater than $0.2$ and returns a maximum matching in over $90\%$ of the $64\times64$ graphs with edge densities less than $0.2$. nDSM performs much better and returns a maximum match in over $97.5\%$ of the cases. Further, as shown in Figs. $11$(b), $11$(d), and $11$(f), in almost all instances these algorithms return a matching that is only one less than the perfect match. This implies almost negligible impact on the overall throughput as also illustrated in Fig. $9$. The graphs in Fig. $11$ also imply a correlation between the density of the graph, the number of vertices in the graph and the difficulty of identifying a maximum size match. In graphs with high edge densities, multiple matches of maximum size exist and it is relatively easy to identify a maximum match. In graphs with low edge densities, the search space is small and this strategy of prioritizing ports with low degrees pays off. However, when the edge densities are neither high nor low, matching in order of increasing degrees does not always lead to a maximum match.

## VI. Conclusion

Input-queued switches based on the virtual-output-queueing (VOQ) architecture are currently the most attractive design for high-performance switches and routers. Past research has concentrated on devising scheduling algorithms that achieve $100\%$ throughput. In contrast, this paper proposes a class of algorithms that concentrate on achieving a low average delay while still achieving near-maximum throughput.

The algorithms presented in this paper are motivated by the observation that current scheduling algorithms are biased in favor of ports with large degrees and cause larger delays at ports with lower degrees. In contrast, our algorithms prioritize the lower degree ports by matching them first.

We present three algorithms that sort the ports in increasing order of degrees and successively match the lowest degree port. The first algorithm, called the *Degree-Sequenced Matching* (DSM) algorithm, matches the lowest degree port to one of its randomly selected neighbors. An improved version of this algorithm matches the lowest degree port with its lowest degree neighbor. The third algorithm, called the *neighbor-aware Degree-Sequenced Matching* (nDSM), takes into account not just the degree of the ports but also the destination ports to which packets are queued. We show, via simulations using both real gateway traffic traces as well as synthetic traffic, that these algorithms achieve a low average delay with negligible loss in throughput as compared to other scheduling algorithms. We also show that these algorithms almost always achieve the maximum match. In the few instances where they do not achieve the maximum match, the match returned is only one less than the maximum possible match.

Although this class of algorithms offers high throughput and low latency close to that of the Maximum Size Matching (MSM) algorithm, it also inherits the fairness problems that are associated with MSM. Specifically, some flows that originate from and are destined to ports with a high degree may be starved indefinitely. One example of such a traffic pattern is a 2-input 2-output switch with one of the input ports always having packets destined to both the output ports and the other input port having packets destined to only one output port. Investigation and improvement of the fairness properties of DSM remains an open problem. Modified versions of DSM, that are based on additional topological properties of bipartite graphs or that consider information about the traffic, such as the queue lengths or the delays of the enqueued packets, have the potential to closely approximate the fairness and stability guarantees of maximum weight matching algorithms.

## References

Anderson, T. E., S. S. Owicki, J. B. Saxe, and C. P. Thacker (1993, Nov.). High-speed switch scheduling for local-area networks. *ACM Trans. Comput. Syst. 11*(4), 319–352.

Angluin, D. and L. G. Valiant (1977, May). Fast probabilistic algorithms for hamiltonian circuits and matchings. In *ACM STOC*, Boulder, CO, USA, pp. 30–41. ACM Press.

Charny, A., P. Krishna, N. Patel, and R. Simcoe (1998, May). Algorithms for providing bandwidth and delay guarantees in input-buffered crossbars with speedup. In *Proc.* 6*-th Int'l Wksp. on Quality-of-Service (IWQoS 98)*, Napa, CA, USA, pp. 235–244.

Gura, N. and H. Eberle (2002, Apr.). The least choice first (lcf) scheduling method for high-speed network switches. In *Proc. IEEE Intl. Parallel and Distributed Processing Symposium*, Ft. Lauderdale, FL, USA, pp. 51–60.

Hopcroft and Karp (1973, Dec.). An $n^{\frac{5}{2}}$ algorithm for maximum matchings in bipartite graphs. *SIAM J. on Computing 2*(4), 225.

Javidi, T., R. Magill, and T. Hrabik (2001, Jun.). A high-throughput scheduling algorithm for a buffered crossbar switch fabric. In *Proc. IEEE Int'l Conf. Communications*, Volume 5, Helsinki, Finland, pp. 1586–1591.

Kam, A. and K.-Y. Siu (1999, Jun.). Linear complexity algorithms for QoS support in input-queued switches with no speedup. *IEEE J. Select. Areas Commun. 17*(6), 1040–1056.

Karol, M., M. Hluchyj, and S. Morgan (1987, Dec.). Input verses output queueing on a space division packet switch. *IEEE Trans. Commun. COM-35*, 1347–1356.

Keslassy, I., R. Zhang-Shen, and N. McKeown (2003, Oct.). Maximum size matching is unstable for any packet switch. *IEEE Commun. Lett. 7*(10), 496–498.

Leonardi, E., F. Neri, and B. Yener (1999, Dec.). Algorithms for virtual output queued switching. In *Proc. IEEE GLOBECOM*, Volume 2, Rio de Janeiro, Brasil, pp. 1203–1210.

McKeown, N. (1999, Apr.). *i*SLIP: A scheduling algorithm for input-queued switches. *IEEE/ACM Trans. Networking 7*(2), 188–201.

McKeown, N., A. Mekkittikul, V. Anantharam, and J. Walrand (1999, Aug.). Achieving 100% throughput in an input-queued switch. *IEEE Trans. Commun. 47*(8), 1260–1267.

Ni, N. and L. N. Bhuyan (2002, Jun.). Fair scheduling in internet routers. *IEEE Trans. on Comput., Special Issue on Quality of Service Issues in Internet Web Services 51*(6), 686–701.

NLANR Colorado State University *OC3c* tap (2003, Nov.). *COS-1069192584-1.tsh.gz.* NLANR Colorado

State University *OC3c* tap. [Available on Request.].

Shreedhar, M. and G. Varghese (1996, Jun.). Efficient fair queueing using deficit round-robin. *IEEE/ACM Trans. on Networking 4*(3), 275–285.

Tamir, Y. and G. L. Frazier (1988, May). High-performance multi-queue buffers for vlsi communications switches. In *Proc.* 15*-th Annual Int'l Symp. on Comp. Arch. (ISCA)*, Honolulu, Hawaii, USA, pp. 343–354.

Zhang, X. and L. Bhuyan (2003, May). Deficit round robin scheduling for input queued switches. *IEEE J. Select. Areas Commun., Special Issue on High Performance Optical/Electronic Switches/Routers for High Speed Internet 21*(4), 584–594.