Integrated Scheduling of Unicast and Multicast Traffic in an Input-Queued Switch

Matthew Andrews Sanjeev Khanna Krishnan Kumaran
Bell Laboratories
600-700 Mountain Avenue
Murray Hill, NJ 07974
{andrews,sanjeev,kumaran}@research.bell-labs.com

Abstract— **We consider the problem of scheduling packets in an input-queued switch when both unicast and multicast traffic is present. In contrast to current approaches which mostly isolate unicast from multicast, we propose an *integrated scheduling procedure* that packs unicast cells into idle slots left by the multicast schedule. While the optimal integrated schedule can be shown to be NP-hard to obtain, we propose both off-line and on-line algorithms with strong theoretical guarantees to perform integration efficiently. Simulations suggest significant improvement in switch throughput using the integrated schedule.**

To further reinforce the importance of performing integration, we study the multicast scheduling problem with and without fanout splitting. Again, we prove hardness of the problem and several of its variants, and propose competitive algorithms. The hardness of multicast scheduling hence emphasizes the importance of integrated scheduling for switch performance.

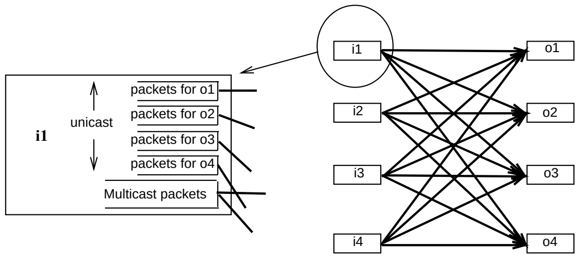

Fig. 1. A schematic of a 4 × 4 switch with unicast and multicast queues.

I. INTRODUCTION

As the demand for bandwidth continues to increase, it is important that we face the challenge of increasing the speed of network elements. Fiber technology now provides us with the ability to transmit data at extremely high rates. As fiber capacity continues to grow, switch contention increasingly becomes a bottleneck that limits performance.

In one model of switch design the packets are buffered at the *outputs* of the switches. This architecture leads to a simple scheduling problem, namely which packet should be transmitted next along the output link. However, in order to implement this output-buffered architecture, the switch fabric of an $N \times N$ switch must operate N times faster than the link speed. This is difficult when the links are fast and the switches are large.

An alternative is to buffer the packets at the *inputs* of the switch (see Figure 1). This makes the implementation easier since now the switch need only operate at the same speed as the links. However, the scheduling problem becomes more difficult since a packet must now contend with the packets that share the same output *and* the packets that share the same input. Nevertheless, the implementational simplicity of input-queued switches means that they are receiving an increasing amount of study. (See Section I-B.)

In this paper we consider the problem of scheduling cells in an input-queued switch when multicast traffic is present. Many broadcast and conferencing applications require a multicast implementation in order to operate efficiently. From the perspective of switch design, multicast traffic consists of packets that arrive at an input and then need to be transmitted to more than one output. One possible solution is to treat a multicast packet as a number of separate unicast packets and schedule them accordingly. However, this is wasteful since the packet could be transmitted to all its outputs simultaneously. We would like to use *multicast scheduling* to take advantage of this fact.

Since most efficient algorithms for unicast scheduling do not apply to multicast, one approach is to isolate the multicast traffic, perhaps by using a separate switching fabric. However, this could lead to ports being idle for some of the time since it is not always possible to select a set of unicast packets that utilize all ports. The problem is exacerbated when there is not enough multicast traffic to utilize a separate fabric. The solution we propose is to allow the unicast traffic to use ports when the multicast schedule leaves them idle. Using this technique we hope to increase the utilization of the switch.

In this paper we study this problem of *integrating* the unicast traffic and the *multicast* traffic. Specifically, we design efficient algorithms which overlay unicast traffic on any given schedule for multicast traffic. We also design an algorithm for the multicast scheduling problem itself. Finally, we study the complexity of obtaining optimal solutions to these problems, and establish their computational intractability. In what follows, we describe in detail our results and some relevant previous work.

A. Results

We assume that each packet consists of a set of cells that must be transmitted to the outputs. In the *on-line* problem the packets arrive over time and the algorithm does not know about arrivals in advance. In the *off-line* problem all packets are present at time 0. We say that an on-line algorithm A is c-competitive if at all times the number of cells scheduled by the optimum algorithm OPT is at most c times the number of cells scheduled by A. Multicast scheduling is termed fanout-splitting or non-fanout-splitting depending on whether or not a packet can be transmitted to its outputs at different times. We return to these definitions in more detail in Section II.

• In Sections III and IV we consider the problem of integrating a unicast schedule with a multicast schedule. Our main result is a 2-competitive algorithm for the on-line problem. We then show that the off-line problem can be formulated as a well-known combinatorial problem known as 3-DIMENSIONAL-MATCHING. Using algorithms for this 3-D MATCHING problem, we are able to devise an approximation algorithm for the off-line problem.
• In Section V we provide evidence that the integration problem is difficult in general. In particular, we show that given a multicast schedule and a set of unicast cells that must be integrated with this schedule, then it is NP-hard to determine the minimum amount of time that it takes to achieve the integration.
• In Section VI we examine the hardness of the multicast problem itself. In particular, we show that the problem is NP-hard for both the fanout-splitting case and the non-fanout-splitting case. Moreover, when no fanout-splitting is allowed, it is intractable even to obtain weak approximations to the optimal solution. This motivates us to examine the fanout-splitting case in more detail. We study this problem in Section VII. Our first result here is that any online algorithm that considers only bounded number of entries in the multicast queue at each input, cannot yield any nontrivial performance guarantees. Next we design a 3-competitive algorithm that operates under the assumption that the online algorithm can examine and schedule any cell in a multicast queue associated with any input. We emphasize here that the hardness of the multicast problem reinforces the importance of the integration problem studied above. An efficient utilization of the switch is achieved by filling up the "holes" in the multicast schedule with unicast cells.
• In Section VIII we carry out a simulation study of the integration problem. In particular, we are interested in the amount of multicast that can be supported, whilst still allowing us to utilize the switch fully. In particular, we are interested in the *schedulable region* of ordered pairs of the form (m, u), where m represents the total load due to multicast cells and u represents the total load due to unicast cells.

We remind the reader that with unicast traffic only we can obtain optimal algorithms that obtain 100% throughput. This is explained by the following fact which is an immediate consequence of a theorem of Birkhoff [4].

Fact 1: Suppose that at most x cells are queued at each input and at most x cells are destined for each output. We can schedule these cells so that they all cross the switch in x steps.

B. Previous work

The problem of scheduling *unicast* cells in an input-queued switch has received much attention. In an early paper, Karol, Hluchyj and Morgan [12] studied a switch design with FIFO queues at the inputs. They showed that the switch can saturate at a utilization of 58.6%. However, this low figure is largely due to a phenomenon called *head-of-line* blocking (HOL) in which the cell at the head of the FIFO queue blocks other cells whose outputs are idle. In consequence, techniques to ease this problem were later proposed in order to obtain higher utilizations. For example, queues in which the FIFO constraint is relaxed are used in [5], [8], [11]. A different approach uses a separate queue for each input-output pair [2], [10], [16], [14], [15], [17], [18], [24], [3]. In particular, McKeown, Anantharam and Walrand [14] showed how to obtain 100% throughput in a single switch using a maximum weight bipartite matching algorithm. A simpler algorithm was later proposed by Mekkittikul and McKeown [16]. For *networks* of input-queued switches where cells must cross more than one switch, Andrews and Zhang [3] have shown how to obtain 100% throughput.

A number of recent papers have concentrated on the problem of emulating an output-queued switch using a combined input and output queued switch that has a "speedup" with respect to the link speed. Chuang, Goel, McKeown and Prabhakar [6] have shown that a speedup of 2 is sufficient for the emulation of many scheduling disciplines for output queueing, e.g. Weighted Fair Queueing. Similar results were obtained by Stoica and Zhang [23]. We note here that these emulation techniques do not apply to either the integration problem or the multicast scheduling problem. The issue of fairness in input-queued switches was also considered by Lund, Phillips and Reingold [13].

The problem of scheduling *multicast cells* in an input-queued switch has received less study. Ahuja, Prabhakar and McKeown [1] presented algorithms for the fan-out splitting problem where there are FIFO queues at the inputs.

A related problem is multicast scheduling in Wavelength Division Multiplexing (WDM) optical networks. See for example Rouskas and Ammar [21] and Ortiz, Rouskas and Perros [19].

II. MODEL AND DEFINITIONS

We consider an $M \times N$ switch with M inputs and N outputs. (See Figure 1.) We aim to schedule *unicast packets* and *multicast packets*. Each unicast packet consists of a single *cell* that must be transmitted from an input i to an output j. Each multicast packet consists of up to N cells. All of these cells must be transmitted from the same input i. However, the destinations of these cells will be distinct outputs. At each time step the cells that we schedule must satisfy the following *scheduling constraint*.

Scheduling constraint. At each time step, for each input we can schedule cells from at most one packet. We can schedule at most one cell to each output.

We note that if a unicast cell is scheduled from an input then it is the *only* cell scheduled from that input. In contrast, many multicast cells may be scheduled from the same input but they must all be part of the same packet. Since each unicast packet consists of a single cell, we use the terms "unicast packet" and "unicast cell" interchangeably.

In this paper we consider two problems, the unicast integration problem and the multicast scheduling problem.

• **Unicast integration problem.** Suppose that the multicast scheduling discipline is predetermined. Due to the inherent difficulty of the multicast scheduling problem, it is likely that during many time steps, some of the inputs and outputs will be not be used by multicast cells. (See Figure 2.) In the integration problem we aim to use this extra switch capacity for unicast cells. More formally, we assume that at each time step we are given the set of input and output ports that are *eligible* for unicast traffic. In addition, a set of unicast packets arrive at the switch over time. Our goal is to schedule the unicast cells as efficiently as possible using the available ports.
• **Multicast scheduling problem.** We also consider the prob-

Time	1	2	3	4	5
o1		i3	i4		i1
o2	i4			i2	
o3		i3			i1
o4	i4		i4	i2	

Fig. 2. An example multicast schedule. The rows correspond to output ports and the columns correspond to time steps. A label in a slot implies that a multicast packet is scheduled there. The label denotes the input port where the packet arrived. An empty slot denotes a place where a unicast packet can be "squeezed" in.

lem of scheduling the multicast cells. There are many natural variants of this problem. With *fanout-splitting* we are allowed to schedule the cells that make up a multicast packet at different times. In the case of *non-fanout-splitting* all these cells must be scheduled at the same time. The problem also depends on the nature of the multicast queue. If the multicast queue is FIFO then we can only schedule the cells from the packet at the head of the queue. We also consider the more general problem in which we are allowed to schedule cells from an arbitrary packet in the queue.

Performance measures. For the unicast scheduling problem, one common measure is *absolute throughput*. For example, if a scheduling algorithm gives 100% throughput then, as long as packets that require any input or output arrive at the switch with rate at most the link speed, all packets are scheduled within a bounded amount of time.

However, for both the integration problem and the multicast scheduling problem, absolute throughput is not the most appropriate measure. This is because the nature of the multicast traffic may mean that even an optimal algorithm cannot achieve 100% throughput. Hence a more appropriate measure is to compare the performance of a candidate algorithm with the performance of the *optimal* algorithm. We do this in slightly different ways depending on whether we are presenting a *positive* result, in which we say what *is* achievable, or a *negative* result, in which we say what is *not* achievable.

For positive results, we typically measure throughput using the concept of *competitive analysis*, first introduced by Sleator and Tarjan [22]. Suppose that at time 0, all the queues are empty. Let OPT be the algorithm that schedules the maximum number of cells after t time steps. We say that a candidate algorithm A is *c-competitive* if, for all t, OPT has scheduled at most c times as many cells as A after t time steps. Clearly a 1-competitive algorithm is optimal. We note that we make no assumptions on the arrival process. The above relation must hold for *any* sequence of arrivals. Moreover, we allow OPT to know all of the arrivals in advance. In contrast, we require algorithm A to be *on-line*, i.e. it cannot know about arrivals in advance.

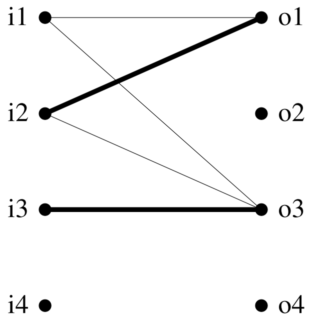

Fig. 3. An example bipartite graph B. A maximum matching is shown in bold.

For our negative results, we typically consider an *off-line* scenario in which all the packets that need to be routed are present at the switch at time 0. For these results we consider both the throughput, where we wish to schedule the maximum number of packets by time t, and the *make-span*, where we wish to schedule all the packets in the minimum time. Note that scheduling in this off-line setting is easier than on-line scheduling since an algorithm can know about all the packets in advance. Hence, if it is hard to optimize for either throughput or make-span in the off-line setting then we cannot hope to have a optimal on-line algorithm. We will say that an off-line minimization (maximization) problem can be approximated to within a factor of α to mean that one can find in polynomial time a solution whose value is at most (at least) α times that of an optimal solution.

III. ON-LINE ALGORITHM FOR THE UNICAST INTEGRATION PROBLEM

In this section we present an on-line 2-competitive algorithm for the unicast integration problem, which constitutes our main result for this problem. As shown in section V, optimal integration is a hard problem, and our algorithm provides a solution that is within a factor of 2 of optimal.

*Algorithm definition (*MATCH*)*

For each time step t do the following.
1. Create a bipartite graph, B, in which the nodes on the left correspond to the inputs and the nodes on the right correspond to the outputs. Put an edge between input node i and output node j if and only if input i and output j are both eligible at time t and the queue for input i and output j is non-empty at time t. (See Figure 3.)
2. Find a maximum matching in the bipartite graph. (This corresponds to the maximum set of cells that can be scheduled at time t.)
3. If the edge between input node i_0 and output node j_0 is in the matching then schedule a unicast cell from input i_0 to output j_0.

Lemma 2: The on-line algorithm MATCH is 2-competitive for the unicast scheduling problem.

Proof: Let D_t^{MATCH} be the number of cells scheduled by MATCH at time t. Let $D_t^{\text{OPT}\setminus\text{MATCH}}$ be the number of cells scheduled by OPT at time t *that* MATCH *has not scheduled by time* t. If MATCH is not 2-competitive then for some t,

$$\sum_t D_t^{\text{OPT}\setminus\text{MATCH}} > \sum_t D_t^{\text{MATCH}}.$$

Hence there exists a time step $t_0 \leq t$ such that,

$$D_{t_0}^{\text{OPT}\setminus\text{MATCH}} > D_{t_0}^{\text{MATCH}}.$$

In other words, there is a set of cells that MATCH did not schedule which MATCH could have scheduled at time t_0. Moreover, this set of cells has larger cardinality than the set of cells that MATCH *did* schedule at time t_0. Hence, when MATCH was considering time step t_0, it did not find the maximum matching in the bipartite graph B. This is a contradiction. ■

In some settings, finding a maximum matching in the second step of the algorithm may be too expensive. If however, we find a matching that is within a factor of α of optimal then the resulting algorithm is 2α-competitive.

IV. ALTERNATIVE FORMULATION OF THE UNICAST INTEGRATION PROBLEM

In order to understand the integration problem better, we now show that in the off-line setting it can be viewed as an instance of the well-known optimization problem called 3-DIMENSIONAL MATCHING. Using this formulation, we give a $(3/2 + \epsilon)$-approximation for the off-line integration problem. Recall that for the off-line problem, all the packets to be scheduled are present initially. We are given a multicast schedule that determines the time steps at which ports are eligible for unicast traffic. (See Figure 2).

The 3D-matching problem is as follows: Given a set $S \subseteq X \times Y \times Z$ where $X, Y,$ and Z are disjoint, find a largest size *independent* subset S' of S, where a subset S' is independent if and only if no two elements in S' agree on any coordinate. This problem is known to be NP-hard. However, for any $\epsilon > 0$, there exists a polynomial time algorithm that gives a $(3/2 + \epsilon)$-approximation to this problem [9].

Starting with an instance of the integration problem on an $M \times N$ switch, we construct an instance of 3D-matching as follows. Let T be the set of time steps at which ports are eligible. (For the off-line problem we can assume that $|T|$ is finite.) Let I and O be the set of inputs and outputs of the switch respectively. Let C be the set of unicast cells that must be scheduled. We define $X = T \times I$, $Z = T \times O$ and $Y = C$. The collection S of tuples is constructed as follows. For each input-output port-pair (i, j) let $c \in C$ be a cell that wishes to go from input i to output j. For each time step $t \in T$ for which input i and output j are both eligible, add a tuple $s = \langle\langle t, i\rangle, c, \langle t, j\rangle\rangle$ to the set S. This completes our construction. The following two lemmas show that in fact, solving the original instance is equivalent to solving this new instance for the 3D-matching problem.

Lemma 3: Let C' be a subset of unicast cells that can be scheduled in the available time slots. Then it can be translated into a solution S' of size $|C'|$ for the 3D-matching instance constructed above.

Proof: Take a cell $c \in C'$ that is scheduled at time step t. Let i and j denote the input and output ports associated with c. Then add the tuple $s_c = \langle\langle t, i\rangle, c, \langle t, j\rangle\rangle$ to S'. Repeat this for every cell in C'. We claim that S' forms an independent collection. Suppose not, then there exist two tuples s, s' that agree on either the first or the third coordinate (they can never agree on the middle coordinate). Without loss of generality, assume that it is the first coordinate that they agree on. This means that two cells in C' are scheduled at the same time from the same input port. But that is impossible since C' defines a valid schedule. ■

Lemma 4: Any solution S' to the 3D-matching instance constructed above can be translated into a subset C' of size $|S'|$ of unicast cells that can be scheduled in the available time slots.

Proof: Take a tuple in S', say $s_c = \langle\langle t, i\rangle, c, \langle t, j\rangle\rangle$. Add the cell c to C' as being scheduled at time t. Repeat for every tuple in S'. We claim that C' defines a valid schedule of size $|S'|$. First observe that since S' is independent, C' contains only a unique copy of any cell. Now suppose there are 2 cells c_1, c_2 in C' that are scheduled at the same time from the same input port (output port). Then it contradicts the independence of S' since the corresponding tuples agree on the first coordinate (third coordinate). The lemma follows. ■

Again, while the above argument establishes that the off-line unicast integration problem can be represented as 3D-matching, the reverse is not true, i.e. not every 3D-matching instance is a unicast integration problem. However, existing results on 3D-matching [9] permit us to obtain an approximation algorithm for off-line unicast integration. We hence obtain the following theorem from prior work.

Theorem 5: [9] For every $\epsilon > 0$, there exists a polynomial time algorithm that computes a $(3/2 + \epsilon)$-approximation to the off-line unicast integration problem.

V. HARDNESS OF THE INTEGRATION PROBLEM

We now show that integrating the unicast traffic with the multicast traffic optimally is NP-hard. Although in Section IV we showed that our problem can be reduced to 3D-matching, for our hardness proof we find it more convenient to reduce the well-known NP-complete problem called 3SAT to our problem.

Theorem 6: The integration problem is NP-hard.

Proof: Let I be an instance of 3SAT with variable set $\{x_0, x_1, \ldots, x_{n-1}\}$ and boolean clause set $\{c_0, c_1, \ldots, c_{m-1}\}$. Each clause contains at most 3 literals. In the 3SAT problem we wish to know if there is a truth assignment to the variables for which all clauses are satisfied. Without loss of generality we assume that for some d each literal x_i and $\bar{x}_i$ appears exactly d times. We also assume that each variable appears in each clause at most once. We construct an instance of the integration problem with a switch of size $(m + 3nd) \times (m + nd)$ and $m + 6nd$ unicast cells.

These cells are defined as follows. In what follows we use $(k + 1)$ to denote $(k + 1) \bmod d$. For each clause c_j we have a unicast cell, C_j, that wishes to go from input j to output j. For each variable x_i we have $6d$ cells,

$$X_i[1], Y_i[1], Z_i[1], \bar{X}_i[1], \bar{Y}_i[1], \bar{Z}_i[1], \ldots,$$
$$X_i[d], Y_i[d], Z_i[d], \bar{X}_i[d], \bar{Y}_i[d], \bar{Z}_i[d].$$

The key correspondence that we enforce between the 3SAT instance I and the scheduling problem is the following. If, for any k, cell $X_i[k]$ is scheduled at time $3i$ then the variable x_i is TRUE. If, for any k, cell $\bar{X}_i[k]$ is scheduled at time $3i$ then the variable x_i is FALSE. We must ensure that there do not exist k, k' such that both $X_i[k]$ and $\bar{X}_i[k']$ are scheduled a time $3i$.

Suppose that the kth appearance of x_i is in clause c_j and the kth appearance of $\bar{x}_i$ is in clause $c_{j'}$. The input and output ports for the above $6nd$ cells are,

	Input	Output
$X_i[k]$	$m + 3id + k$	j
$Y_i[k]$	$m + (3i+1)d + k$	j
$Z_i[k]$	$m + (3i+1)d + k$	j'
$\bar{X}_i[k]$	$m + (3i+2)d + k$	j'
$\bar{Y}_i[k]$	$m + (3i+2)d + k$	$m + id + k$
$\bar{Z}_i[k]$	$m + 3id + (k+1)$	$m + id + k$

It remains to define which time slots are eligible for the unicast cells, i.e. which cells are *not* being used by multicast traffic. If clause c_j contains variables $x_{i_1}, x_{i_2}, x_{i_3}$ (or their negations) then input j is eligible for unicast cells at times $3i_1 + 1, 3i_2 + 1$ and $3i_3 + 1$, and output j is eligible at times $3i_1, 3i_1 + 1, 3i_1 + 2, 3i_2, 3i_2 + 1, 3i_2 + 2, 3i_3, 3i_3 + 1, 3i_3 + 2$. For $1 \leq i \leq n$ and $1 \leq k \leq d$, inputs $m + 3id + k$ and $m + (3i+2)d + k$ are eligible at times $3i$ and $3i+1$, input $m+(3i+1)d+k$ is eligible at times $3i$ and $3i+2$, and output $m + id + k$ is eligible at times $3i$ and $3i+1$.

It can be verified that in any legal schedule there do not exist k, k' such that both $X_i[k]$ and $\bar{X}_i[k']$ are scheduled a time $3i$. This is because if $X_i[k]$ is scheduled at time $3i$ then $Y_i[k]$ is scheduled at time $3i+2$ which implies that $Z_i[k]$ is scheduled at time $3i$ which implies that $\bar{X}_i[k]$ is scheduled at time $3i+1$ etc. These implications continue for all k'. A similar argument holds if $\bar{X}_i[k]$ is scheduled at time $3i$. The following two lemmas complete the reduction since they imply that an optimal algorithm for the integration problem would enable us to solve the instance of 3SAT.

Lemma 7: Suppose that I is satisfiable. Then all the unicast cells can be scheduled by time $3n$.

Proof: Consider a satisfying assignment. If x_i is TRUE in this assignment then for all k we schedule $X_i[k], Z_i[k]$ and $\bar{Y}_i[k]$ at time $3i$, $\bar{X}_i[k]$ and $\bar{Z}_i[k]$ at time $3i+1$ and $Y_i[k]$ at time $3i+2$. If x_i is FALSE in the satisfying assignment then for all k we schedule $X_i[k]$ and $\bar{Y}_i[k]$ at time $3i+1$, $\bar{X}_i[k], \bar{Z}_i[k], Y_i[k]$ at time $3i$ and $Z_i[k]$ at time $3i+2$. If the clause c_j is satisfied by either x_i or $\bar{x}_i$ then we schedule cell C_j at time $3i+1$. It can be verified that this is a legal schedule. In particular, we can schedule C_j at time $3i+1$ since by assumption, each variable appears at most once in each clause and so c_j cannot contain both x_i *and* $\bar{x}_i$. ■

Lemma 8: Suppose that all the unicast cells can be scheduled by time $3n$. Then I is satisfiable.

Proof: We construct a satisfying assignment as follows. Suppose that C_j is scheduled at time $3i+1$. Then c_j either contains the literal x_i or the literal $\bar{x}_i$. If c_j contains the literal x_i then we set x_i to TRUE. If c_j contains the literal $\bar{x}_i$ then we set x_i to FALSE. The clause c_j is now satisfied. This is a valid assignment so long as we cannot find two clause cells C_j and $C_{j'}$ both scheduled at time $3i+1$ such that c_j contains the literal x_i and $c_{j'}$ contains the literal $\bar{x}_i$. However, if this were the case then there would exist k and k' such that $X_i[k]$ and $X_i[k']$ are both scheduled at time $3i$. This cannot happen by our construction. ■

■

VI. Hardness of Multicast

In this section we prove that the problem of optimally scheduling multicast cells is hard, even if we allow fanout-splitting. We establish the hardness of the multicast problem by using a *reduction* from a fundamental combinatorial problem known as GRAPH COLORING: Given a graph $G(V, E)$, find the least integer k such that there exists a map $f : V \rightarrow [1..k]$ with the property that $f(u) \neq f(v)$ if $(u, v) \in E$. The problem is well-known to be NP-hard. We will use the following negative result, shown recently by Feige and Kilian [7]:

Proposition 9: [7] For any fixed $\epsilon > 0$, GRAPH COLORING can not be approximated to within a factor of $n^{1-\epsilon}$ if NP $\neq$ ZPP.

The reader may refer to the text by Papadimitriou [20] for a definition of the class ZPP.

Theorem 10: The multicast problem without fanout-splitting is NP-hard. Furthermore, it is hard to approximate it within a factor of $n^{1-\epsilon}$ unless NP=ZPP. Here n denotes the number of inputs to the switch.

Proof: Let $G = (V, E)$ be an input to GRAPH COLORING. We construct a multicast problem M_G with the property that M_G can be scheduled in k steps if and only if G can be colored with k colors. The switch in the instance M_G has n inputs, one for each vertex in V, and m outputs, one for each edge in E. At each input we have 1 multicast packet whose outputs are chosen in the following manner. First number the nodes in V from 1 to n and the edges in M from 1 to m. The multicast packet at input i has output j as one of its destinations if and only if the edge with number j is incident to the node with number i.

Now consider any scheduling of these packets in, say, k steps. Assign color $t \in [1..k]$ to node $i \in V$ if and only if it is scheduled at time t. We claim that this is then a valid coloring of G. Suppose not, then there exists $i, i' \in V$ such that $j = (i, i') \in E$ and i and i' were scheduled at the same time t. But this is a contradiction since they both share output j. Hence this is indeed a valid coloring. In the other direction, suppose we are given any k-coloring of nodes of G, here the colors are assumed to be in the set $[1..k]$. Then it is easily seen that scheduling the packet at input node i at time equal to the color assigned to node i, constitutes a valid schedule. Thus G has a k-coloring if and only if M_G can be scheduled in k steps. The NP-hardness now follows from the NP-hardness of GRAPH COLORING, and the $n^{1-\epsilon}$-hardness of approximation follows from Proposition 9. ■

Since a factor n-approximation to the multicast problem can be obtained rather trivially (just schedule one packet at every time step), the above result says that one can not essentially do much better. This hardness is due to the restriction that no fanout-splitting is allowed. Our next result shows that the problem remains NP-hard to solve optimally even when fanout splitting is allowed.

Theorem 11: The multicast problem is NP-hard even with fanout-splitting.

Proof: Our construction is a small modification of the construction used in Theorem 10. Again, let $G = (V, E)$ be an input to GRAPH COLORING and we wish to determine if G can be colored with k colors, for some $k \geq 3$. We construct a multicast problem $M_{G,k}$ with the property that $M_{G,k}$ can be scheduled in k steps if and only if G has a k-coloring. The switch in the instance $M_{G,k}$ has n inputs, one for each vertex in V, and $m + n$

outputs, one for each edge in E and one for each vertex in V. At each input we have 1 multicast packet and $k-1$ unicast packets. Each unicast packet at input i is destined for output $m+i$. We choose the outputs for the multicast packet in the following manner (similar to Theorem 10). First number the nodes in V from 1 to n and the edges in M from 1 to m. The multicast packet at input i has output j as one of its destinations if and only if the edge with number j is incident to the node with number i.

Suppose that all the packets are scheduled in less than $k+1$ time steps. Since there are k packets at each input then they must be scheduled in exactly k time steps. Moreover, each multicast packet must be completely scheduled in one time step, i.e. no fan-out splitting is possible. Let t be the time step at which the multicast packet at input i is scheduled. Color $i \in V$ with color t. Now suppose that $j = (i, i')$ is an edge in E. Then the multicast packets at inputs i and i' both have an output packet destined for output j. Hence these multicast packets are scheduled at different times which implies that i and i' are assigned different colors. Thus we have a valid k-coloring of the graph G.

To see the other direction, suppose that G is colorable with the k colors $\{1, 2, \ldots, k\}$. We construct the following schedule for packets. If node $i \in V$ is colored with color t then we schedule the multicast packet at input i at time step t. We schedule the unicast packets at input i at time steps $1, \ldots, t-1, t+1, \ldots, k$. It remains to show that we never schedule two packets for the same output simultaneously. This is clear for the unicast packets since the unicast packets for distinct inputs are destined for distinct outputs. Now suppose that two multicast packets from inputs i and i' are both scheduled at time t and they share an output j. Then, by construction we know that we have an edge $(i, i') \in E$. In addition, the nodes i and i' in V are both colored with color t in the k-coloring. This is a contradiction.

Since it is NP-hard to determine if a graph is k-colorable for any $k \geq 3$, our result follows. ■

A useful corollary of the preceding theorem is as below.

Corollary 12: For any fixed $\epsilon > 0$, it is NP-hard to approximate multicast scheduling to within a factor of $\frac{4}{3} - \epsilon$.

Proof: An immediate consequence of Theorem 11 and the fact that it is NP-hard to determine if a graph G is 3-colorable. ■

VII. Online Scheduling of Multicast Packets with Fanout Splitting

In this section, we examine the problem of scheduling multicast packets in an online setting. When no fanout splitting is allowed, Theorem 10 shows that it is hard to beat the trivial scheduling policy even in an off-line setting. Thus it is natural to allow fanout splitting when studying the question in an online setting. Let n be the number of inputs and outputs of the switch. Our first result is a lower bound of $\Omega(n)$ on the competitive ratio of any online algorithm that schedules only by looking at a bounded number of packets in the FIFO queue at each node. Again, it implies that the trivial guarantee is essentially the best possible under this restriction. Thus to do any better, we must allow the online algorithm to examine an unbounded number of packets at each input queue. Our second result is a 3-competitive algorithm that works under the assumption that any number of packets waiting at an input node can be examined and scheduled in an arbitrary order.

Theorem 13: For any fixed positive integer α, an online algorithm that schedules by examining only the leading α packets in each multicast queue, has a competitive ratio of $\Omega(n)$ even when the online algorithm is allowed fanout-splitting while the optimal algorithm is allowed no fanout-splitting.

Proof: We illustrate our construction only for the case of $\alpha = 1$, and later sketch the extension to general α. We assume a multicast switch with n inputs and n outputs. Let A be any online algorithm and let O denote the adversary as well as its algorithm. We will construct an instance where in a window of size $n/2$, A will schedule no more than $n/2$ packets while O, during the same window, will schedule $\Omega(n^2)$ packets with no fanout-splitting, thus giving us the desired gap.

The leading packet at each input queue seeks to go to output ports $\{1, 2\}$. Whenever A schedules an input packet at some input port i, the next packet in the queue at node i is also revealed to be for output ports $\{1, 2\}$. Thus after $n/2$ time steps, A has scheduled exactly $n/2$ packets. Assume w.l.o.g. that the input ports whose packets A schedules in the first $n/2$ time steps are $1, 2, ..., p$ for some $p \leq n/2$. In contrast, during these $n/2$ time steps, O schedules packets waiting at input ports $n/2+1, ..., 3n/4$. The input queue at node i where $i \in [n/2+1..3n/4]$, looks as follows:

$$\{1, 2\}, \underbrace{\{i - n/2, i\}, \{i - n/2, i\}, ..., \{i - n/2, i\}}_{\frac{n}{4} \text{ times}}.$$

In the first $n/4$ time steps, O schedules the leading packet at each input port $i \in [n/2+1..3n/4]$, this is done one input port at a time. Then in the next $n/4$ time steps, it keeps scheduling one packet at each of the same input ports, this time, in parallel. It is easy to verify that no conflict arises. Thus at the end of the $n/2$ time steps, O would have scheduled $\Omega(n^2)$ packets in contrast to the $O(n)$ packets scheduled by A. This establishes our claimed gap.

In order to extend to arbitrary α, O simply makes the leading α packets at any input queue to seek outputs $\{1, 2\}$ and then an analogous argument applies. ■

Given the negative result above and those of the preceding section, it seems natural to attribute the following additional power to our online algorithm: we allow it to examine every packet waiting at any input queue and schedule any such packet using fanout splitting. Under this assumption, we now present a simple 3-competitive greedy algorithm. This algorithm, referred to as Greedy, works as follows. At each time step, it examines each input in turn. For each packet c waiting at input i, let o_c be the set of outputs to which packet c must still be routed. Let O_{i-1} be the set of outputs to which we have chosen to route cells from inputs $1, \ldots, i-1$. Let c be the packet for which $|o_c \setminus O_{i-1}|$ is maximum. For input i, we route packet c to the outputs in $o_c \setminus O_{i-1}$. We now set $O_i = O_{i-1} \cup (o_c \setminus O_{i-1})$.

Theorem 14: Greedy is 3-competitive for the multicast problem with fanout-splitting.

Proof: Let G be the set of cells serviced by Greedy during the interval $[0, T]$. Let G_t be the set of cells serviced by Greedy at time step t. Let $G_t^{(i)}$ be the set of cells serviced by Greedy from input i at time step t.

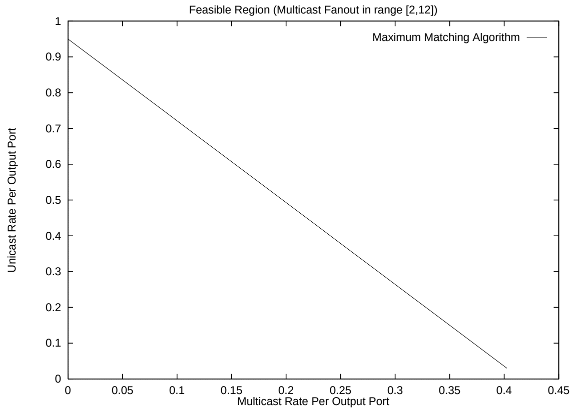

Fig. 4. Feasible region.

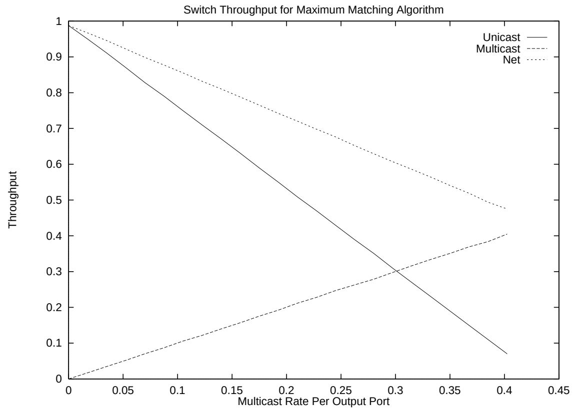

Fig. 5. Throughput as a function of multicast rate.

Now let us consider the cells that are serviced by OPT during the time interval $[0, T]$. We can divide these cells into three types.

- Let A be the set of cells serviced by OPT during $[0, T]$ that are also serviced by GREEDY during $[0, T]$. Clearly $|A| \leq |G|$.
- Let B_t be the set of cells not in A that are serviced by OPT during time step t and that share an output port with a cell in G_t. Clearly, $|B_t| \leq |G_t|$.
- Let $C_t^{(i)}$ be the set of cells serviced by OPT from input i at time t that are not members of either B_t or A. All of the cells in $C_t^{(i)}$ are candidates for GREEDY to service when it considers input i at time t. Hence $|C_t^{(i)}| \leq |G_t^{(i)}|$.

It is clear that the above three types of sets include all the cells serviced by OPT. Hence the number of cells serviced by OPT during the interval OPT is at most,

$$\begin{aligned} |A| + \sum_t |B_t| + \sum_{i,t} |C_t^{(i)}| &\leq |G| + \sum_t |G_t| + \sum_{i,t} |G_t^{(i)}| \\ &= |G| + |G| + |G|. \end{aligned}$$

■

VIII. Numerical Results

We now present simulation results for the unicast integration algorithm described in Section III. We consider a 16x16 input queued switch supporting homogeneous unicast and multicast traffic at each input port. Since the main thrust of our work is to pack unicast traffic onto the idle time-slots left over by the multicast, we restrict ourselves to the following simple round robin non-fanout-splitting scheduling algorithm for the multicast.

- At each time step t, the multicast scheduler begins by scheduling the head of input port $(i + 1) \bmod N$, where i denotes the starting port of the schedule at $t - 1$.
- The scheduler continues by examining the head of successive input ports $i + 2, i + 3, \ldots$ For each port, the packet at the head is scheduled if all of its destination output ports are free.

As described in Section III, the unicast cells are then scheduled by obtaining a maximum matching between the unscheduled input and output port pairs.

To obtain the operating region of the switch, one must determine the maximum unicast rate that can be supported as a function of multicast rate. However, this requires testing the queues for stability, which is known to be a difficult task from queueing theory. We devised the following test, which proved to be somewhat conservative, but nevertheless gave fairly consistent results.

- Maintain an exponentially smoothed moving average of the quantity

$$m_t = \frac{q_t}{T_t}$$

where q_t is the queue size at time t and T_t is the total number of packets served until t. Thus, we have

$$M_t = \mu m_t + (1 - \mu) M_{t-1}$$

as our convergence criterion, where $\mu \in [0, 1]$ is a smoothing parameter which must be small in order to obtain a large averaging window.

- For a stable queue, $M_t \to 0$ as $t \to \infty$. In our simulation, we consider a queue to be stable if M_t is small after a long time.

The simulations were carried out for a 16x16 input-queued switch with homogeneous loading on all ports. Homogeneity here means equal arrival rates for unicast cells for each input-output port pair and equal multicast rates for each input port. Further, the fanout of the multicast cells was chosen uniformly in the range $[2, 12]$, with each output being equally likely. We are hence left with two parameters of interest, namely the unicast and multicast rates per output port. The objective of the simulation was to study the feasible region in this 2-dimensional space and the throughput achieved.

The feasible region of our scheduling algorithm is shown in Figure 4, while the throughput as a function of multicast rate is shown in Figure 5. Notice that the boundary of the feasible region, as well as the throughput, behave linearly for all practical purposes. As seen in Figure 5, the maximum-matching based scheduling algorithm is almost maximally efficient for small multicast rates. While the throughput drops significantly for higher multicast rates, owing to the complexity of multicast scheduling, notice that a significant amount of unicast cells make it through in the switch. For example, when the multicast rate is 0.3 per output port, an equal amount of unicast can be squeezed through leading to twice the throughput as compared to scheduling only multicast. This points to the importance of unicast integration for efficient use of the switch. We do not consider multicast rates greater than about 0.4 per output port, since at that rate the multicast queues themselves become unstable.

While more efficient multicast scheduling could reduce the space available to unicast, we nevertheless expect the gains to remain significant. In particular, owing to the hardness of the multicast problem, we expect most multicast algorithms to allow substantial scope for unicast integration.

IX. Conclusion

In this paper we consider two problems related to the scheduling of multicast packets in an input-queued switch. We first assume that the multicast schedule is given and we present algorithms for scheduling unicast traffic on the idle ports. We believe that this is especially important in view of our later results which show that multicast scheduling is a hard problem. With any practical multicast scheduling algorithm there are likely to be many "holes" in the schedule where extra unicast cells can be "squeezed" in. Our experimental results support this conclusion. We also present a new analysis of the multicast scheduling problem when fanout-splitting is allowed.

One important problem for further work is to try to reduce the competitive ratios in our positive results or else show using lower bounds that these ratios cannot be reduced. Another interesting problem is the queueing behavior of an input-queued switch when it is offered stochastically generated multicast traffic.

References

[1] R. Ahuja, B. Prabhakar, and N. McKeown. Multicast scheduling for input-queued switches. *IEEE Journal on Selected Areas in Communications*, May 1996.

[2] T. E. Anderson, S. S. Owicki, J. B. Saxe, and C. P. Thacker. High-speed switch scheduling for local-area networks. *ACM Transactions on Computer Systems*, 11(4):319 – 352, 1993.

[3] M. Andrews and L. Zhang. Stability results for networks with input and output blocking. In *Proceedings of the 30th Annual ACM Symposium on Theory of Computing*, Dallas, TX, May 1998.

[4] G. Birkhoff. Tres observaciones sobre el algebra lineal. *Univ. Nac. Tucumán Rev. Ser. A5*, pages 147 – 150, 1946.

[5] M. Chen and N. D. Georganas. A fast algorithm for multi-channel/port traffic scheduling. In *Proceedings of IEEE SUPERCOM/ICC*, pages 96 – 100, 1994.

[6] S. Chuang, A. Goel, N. McKeown, and B. Prabhakar. Matching output queueing with a combined input output queued switch. In *Proceedings of IEEE INFOCOM '99*, 1999.

[7] U. Feige and J. Kilian. Zero knowledge and the chromatic number. In *Proceedings of the Eleventh Annual IEEE Conference on Computational Complexity*, pages 278–287, 1996.

[8] A. Huang and S. Knauer. Starlite: A wideband digital switch. In *Proceedings of GLOBECOM*, pages 121 – 125, 1984.

[9] C. A. J. Hurkens and A. Schrijver. On the size of systems of sets every t of which have an SDR, with an application to the worst-case ratio of heuristics for packing problems. *SIAM J. Disc. Math*, 2(1):68 – 72, February 1989.

[10] M. Karol, K. Eng, and H. Obara. Improving the performance of input-queued ATM packet switches. In *Proceedings of IEEE INFOCOM '92*, pages 110 – 115, 1992.

[11] M. Karol and M. Hluchyj. Queueing in high-performance packet-switching. *IEEE Journal of Selected Areas in Communications*, pages 1587 – 1597, Dec. 1988.

[12] M. J. Karol, M. G. Hluchyj, and S. P. Morgan. Input versus output queueing on a space-division packet switch. *IEEE Transactions on Communications*, 35(12):1347 – 1356, December 1987.

[13] C. Lund, S. Phillips, and N. Reingold. Fair prioritized scheduling in an input-buffered switch. *Broadband Communications*, pages 358 – 369, 1996.

[14] N. W. McKeown, V. Anantharam, and J. Walrand. Achieving 100% throughput in an input-queued switch. In *Proceedings of IEEE INFOCOM '96*, pages 296 – 302, San Francisco, CA, March 1996.

[15] N. W. McKeown, P. Varaiya, and J. Walrand. Scheduling cells in an input-queued switch. *Electronics Letters*, 1993.

[16] A. Mekkittikul and N. McKeown. A practical scheduling algorithm to achieve 100% throughput in input-queued switches. In *Proceedings of IEEE INFOCOM '98*, San Francisco, CA, April 1998.

[17] H. Obara. Optimum architecture for input queueing ATM switches. *IEEE Electronic Letters*, pages 555 – 557, March 1991.

[18] H. Obara and T. Yasushi. An efficient contention resolution algorithm for input queueing ATM cross-connect switches. *Int. J. Digital Analog Cabled Syst.*, 2(4):261 – 267, 1989.

[19] Z. Ortiz, G. N. Rouskas, and H. G. Perros. Scheduling of multicast traffic in tunable-receiver WDM networks with non-negligible tuning latencies. In *Proceedings of ACM SIGCOMM '97*, 1997.

[20] C. H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.

[21] G. N. Rouskas and M. H. Ammar. Multi-destination communication over tunable-receiver single-hop WDM networks. *IEEE Journal on Selected Areas in Communications*, 15(3):501–511, April 1997.

[22] D. D. Sleator and R. E. Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28(2):202–208, 1985.

[23] I. Stoica and H. Zhang. Exact emulation of an output queueing switch by a combined input output queueing switch. In *Proceedings of the Sixth IEEE/IFIP International Workshop on Quality of Service*, Napa, CA, May 1998.

[24] Y. Tamir and G. Frazier. High-performance multi-queue buffers for VLSI communication switches. In *Proceedings of the 15th Annual Symposium on Computer Architecture*, pages 343 – 354, 1988.