

Extending the Birkhoff-von Neumann Switching Strategy for Multicast – On the use of Optical Splitting in Switches

Jay Kumar Sundararajan, *Student Member, IEEE*, Supratim Deb, and Muriel Médard, *Senior Member, IEEE*

Abstract—The Birkhoff-von Neumann (BVN) strategy for single-stage input-queued crossbar switches does not support multicast, as it considers only permutation-based switch configurations. This paper extends the BVN strategy to multicast switching, where an input can simultaneously transmit to multiple outputs. Knowledge of the average rates of flows is used to compute an offline schedule. We begin by considering a system in which the fanout of each flow is split in a predecided manner. We call this *static* splitting (as opposed to *dynamic* splitting where no such constraint is imposed), and we study the rate region of the switch under this restriction.

We provide a graph-theoretic formulation of the rate region. Such a formulation enables the use of results and algorithms from graph-theory literature, in the context of switch scheduling. Specifically, we show that the multicast rate region with no fanout-splitting is the stable set polytope of the traffic pattern’s “conflict graph”. This result extends to the static splitting case as well. We construct examples to show that there is no single dominant static splitting strategy in terms of rate region. We show that deciding achievability of a given set of rates is NP -hard for static and dynamic splitting. We also show that, computing the offline schedule with static splitting reduces to fractional weighted graph coloring, which takes polynomial time for perfect graphs. We present several types of traffic patterns whose conflict graphs are perfect. For the more general case of arbitrary traffic patterns, we show that for $K \times N$ switches, where K is a constant with respect to N , fractional weighted coloring of the conflict graph can be performed in time polynomial in the number of flows.

Our study has implications for optical networks in the context of the time-domain wavelength interleaved network (TWIN) approach that was introduced by Ross *et al.* Here the entire optical network is viewed as a single switch, with buffering at the edges and time-division multiplexing of optical circuits in the core. This approach may obviate the need for optical packet header recognition. Inherently, the optical domain enables inputs to transmit to many outputs at once. The static splitting restriction is justified from a practical perspective, as it does not require the network to be reconfigured as often as dynamic splitting does. In addition, it also simplifies the queue management for multicast.

We propose an online algorithm based on our conflict graph approach. Such algorithms do require header recognition. In optical networks, optimal algorithms based on matchings and

extensions thereof may be incompatible with the speed requirements of optical switches. Thus, we envisage simplified algorithms with reduced complexity. We simulate our algorithm along with two other online algorithms – one based on a modification of ESLIP, and the other on a copy-and-use- i -SLIP strategy. Using these simulations, we demonstrate that no static splitting strategy is dominant, even in an online setting in a bigger switch.

Index Terms—Multicast switching, rate region, fanout splitting, stable set polytope.

I. INTRODUCTION

THE problem of switch scheduling is well studied in the literature. It is known that output queued (OQ) switches can achieve 100% throughput for all admissible¹ traffic and can also provide quality of service (QoS) guarantees. However, OQ switches do not scale well with the switch size as they require a large memory bandwidth and a speedup of N in an $N \times N$ switch. An input-queued (IQ) switch architecture, with virtual output queues to avoid head-of-line (HOL) blocking², is a scalable option, since it operates at the line rate. McKeown *et al.* [6] proved that IQ switches can achieve 100% throughput for all admissible unicast arrival patterns and gave a scheduling algorithm known as the maximum weighted matching (MWM) algorithm. This result implies that an IQ switch has the same capacity region as the OQ switch. A detailed survey of the various extensions and simplifications of the MWM algorithm is presented in [1]. A major drawback of MWM-based approaches is that they provably do not provide cell³ delay guarantees. There are several schemes that address this issue. One such scheme is the Birkhoff-von Neumann switch proposed by Chang *et al.* [9].

The Birkhoff-von Neumann (BVN) switch provides 100% throughput for all admissible traffic, and gives deterministic cell delay guarantees for certain types of traffic. This switch is based on a theorem by Birkhoff [2] and von Neumann [3] that, any doubly stochastic matrix⁴ can be expressed as a convex combination of permutation matrices. A permutation matrix naturally corresponds to a valid unicast switch state. Therefore, if the matrix of required rates for every input-output pair is

¹Admissible means no input or output is oversubscribed

²Head-of-line blocking means a packet is blocked even when its destination is free, because the packet at the head of its queue is blocked by a conflict

³Packets arriving at the switch are split into fixed-size units called *cells* which are reassembled into packets at the output.

⁴A *doubly stochastic matrix* is one whose row and column sums are all equal to unity.

Manuscript received March 27, 2006; revised December 19, 2006. This work was supported by: University of Illinois grant 2003-07092-1, NSF grant CNS-0434974 and NSF grant CNS-043974. Parts of this work have appeared as a short paper titled “Extending the Birkhoff-von Neumann Switching Strategy to Multicast Switches” in the *Proceedings of the IFIP Networking 2005 Conference*, May 2-6 2005, Waterloo, Canada.

J. Sundararajan and M. Médard are with the Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, MA 02139, USA (e-mail: {jaykumar, medard}@mit.edu).

S. Deb is a Member of the Technical Staff at Alcatel-Lucent Bell Laboratories, Bangalore, India (e-mail: supratim@alcatel-lucent.com).

Digital Object Identifier 10.1109/JSAC-OCN.2007.026006.

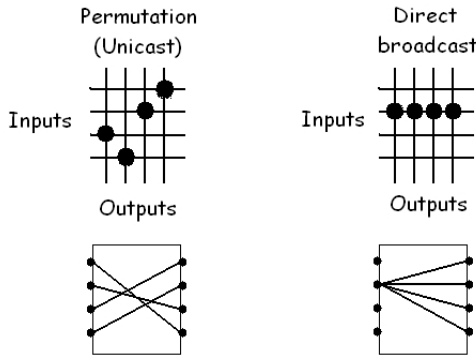


Fig. 1. Switch connection states : Permutation and Direct Multicast

doubly stochastic, we can compute the convex combination, and thereby obtain an offline schedule where a particular permutation state is maintained in the switch for a fraction of time equal to its coefficient in the convex combination.

Such rate decomposition based approaches for offline scheduling have a direct impact on optical networks. For instance, the *time-domain wavelength interleaved network* or TWIN approach, which was proposed in [7] and [8], views the entire network as a single switch, with buffering at the edges and all-optical circuits in the core. Rate decomposition leads to a virtual circuit switching solution which involves time-division multiplexing among virtual circuits. In this work, we extend this idea to the case of multicast flows. We take into account, the intrinsic property of light that it can be transmitted to multiple receivers at once, using power splitters. The impact on optical networks is discussed further in Section I-E.

A. Different Techniques of Multicasting

A natural extension of the unicast scheduling problem is the multicast case, where connections are no longer point-to-point, but may have multiple destinations. Some switching fabrics such as the crossbar and the optical switch have *intrinsic multicast capability*⁵. It is possible to split the light in an optical switch, and send it to multiple outputs at once. Therefore, there are valid switch connection states other than matchings. For instance, direct multicast is possible as shown in Figure 1. In fact, we now have the option of breaking the *fanout* (the destination set) of a multicast cell into subsets and sending the cell to the outputs one subset at a time, across several time-slots. This is called *fanout-splitting*. Note that, these observations are also true in the context of a TWIN-like architecture [7], where the entire network is viewed as a switch and scheduling is done among virtual circuits.

The question that follows naturally is – how to split the fanout of each multicast flow? There are many options (see Figure 2), each requiring a different queuing policy:

- 1) **Copying:** Each input maintains one virtual output queue (VOQ) for every output. When a multicast cell arrives at an input, it is replicated as many times as its fanout size. One copy is added to the VOQ of each destination. This is equivalent to splitting the multicast

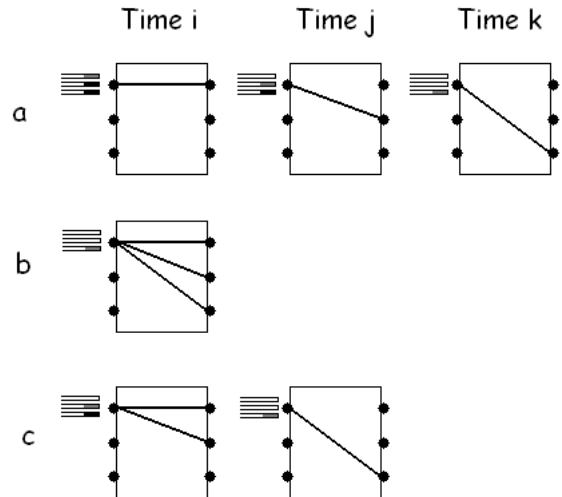


Fig. 2. (a) *Copying*: Cell sent to 3 outputs in 3 time slots (b) *No-splitting*: Cell sent to all 3 outputs in same time slot (c) *Partial Fanout Splitting*: Cell sent to 2 outputs in one slot, and to third output in another slot

into a collection of unicast flows. Thus, the fanout is split completely.

- 2) **No-splitting:** A multicast cell is sent to all its destinations in a single time slot. The fanout is not split at all. Each input maintains a separate VOQ for every multicast flow to avoid HOL blocking. The switch is assumed to have intrinsic multicast capability.
- 3) **Fanout-splitting:** Multiple copies of the multicast cell are generated and, in each slot, one copy is transferred to a subset of the fanout which has not already got the cell. The fanout is thus split partially. This can be implemented in two ways:

Static splitting: Here, the multicast traffic pattern is assumed to be known beforehand. For each flow, the manner in which it will be split is decided offline, and is kept fixed. All cells of a flow are split in the same manner. This is like replacing the original multicast by a set of “split flows”, for which further splitting is not allowed. Each input must maintain a separate VOQ for every split flow. When a multicast cell arrives at an input, it is replicated according to the predecided policy, and one copy is added to the VOQ of each of its split flows.

Dynamic splitting: In dynamic splitting, the restriction of fixing the manner in which the flow will be split is removed. Each cell of a flow may be split in a different way. One way of implementing this strategy is: each input maintains a VOQ for every subset of the fanout. When a multicast cell arrives at an input, it is transferred to some subset of its fanout and then, re-enqueued into the VOQ corresponding to the remaining part of the fanout.

Dynamic splitting clearly subsumes the other options. It gives better throughput than no-splitting [5]. However, this benefit comes at a cost. Since the way flows are split is not known beforehand, the part of the fanout that remains to be served at

⁵The ability to simultaneously transfer a cell to multiple outputs using simultaneous switching paths

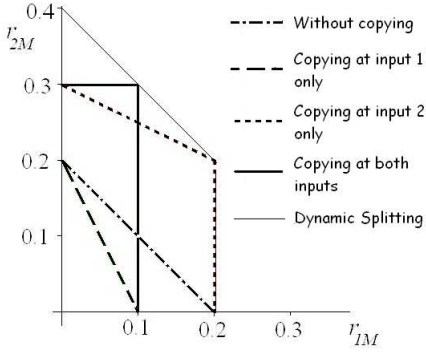


Fig. 3. The rate region of multicast flows in a 2×2 switch with unicast rates fixed as: $r_{11} = r_{12} = 0.4$, $r_{21} = r_{22} = 0.2$ (r_{ij} is the unicast from input i to output j ; r_{iM} is the multicast from input i to both outputs, $i, j = 1, 2$)

some point of time could be an arbitrary subset of the original fanout. As a result, queue management becomes very difficult, even if the traffic pattern is known beforehand. To prevent HOL blocking, each input has to maintain a separate VOQ for every possible subset of the fanout, which results in an exponential number of queues. If we have fewer queues, we need a non-trivial scheme for sharing them among the various flows (see [18]). In contrast, **the static splitting approach requires a much smaller number of queues and much simpler queue management.** At each input, one VOQ for every split flow is sufficient to prevent head-of-line blocking.

Even in an offline virtual-circuit based approach, for instance in optical networks, dynamic splitting can cause practical difficulties. Since there is no restriction on how the flows may be split, one could potentially have a large number of ways of splitting a flow, and the offline schedule could involve a prohibitively large number of configurations. In high-speed optical networks, reconfiguration cannot be done very frequently for practical reasons. This problem is taken care of, if we restrict the ways in which a flow may be split by using static splitting. In this sense, static splitting is more suitable for optical networks – even though we lose some throughput, the implementation becomes simpler.

In the light of the above discussion, we will consider only static splitting in this work. Our goal is to characterize the achievable rate region under the constraint of simple implementation.

B. An Example

We present an example in a 2×2 switch, which elucidates the effect of the fanout-splitting strategy on the rate region. Figure 3 shows the rate region for various fanout-splitting strategies, in terms of achievable multicast rates, for a fixed level of unicast.

First, we describe how to compute these regions. An obvious requirement for all the cases is that all rates must be non-negative. For the no-splitting case, when either multicast is being served, no other flow can be served. Hence, the problem reduces to serving the unicasts during the time that remains after serving the multicasts, namely, a $(1 - r_{1M} - r_{2M})$ fraction of time. Therefore, using the BVN result for unicasts ([2], [3]),

the rate region is:

$$r_{11} + r_{12} \leq (1 - r_{1M} - r_{2M}) \quad (1)$$

$$r_{21} + r_{22} \leq (1 - r_{1M} - r_{2M}) \quad (2)$$

$$r_{11} + r_{21} \leq (1 - r_{1M} - r_{2M}) \quad (3)$$

$$r_{12} + r_{22} \leq (1 - r_{1M} - r_{2M}) \quad (4)$$

The rate regions for the other static splitting strategies can be obtained by replacing the rates in the above inequalities with appropriately modified effective rates. For instance, if copying is used at input 1 alone, then the effective rates become: $r_{1M}^{eff} = 0$, $r_{11}^{eff} = r_{11} + r_{1M}$, and $r_{12}^{eff} = r_{12} + r_{1M}$, while other rates remain the same. This is because effectively, every multicast flow packet that arrives gets copied to the queues of both unicasts from input 1.

To obtain the rate region for dynamic splitting, we note first that, the following two inequalities are necessary, as they correspond to the condition that neither input 1 nor output 1 may be over-subscribed: $r_{11} + r_{12} + r_{1M} \leq 1$ and $r_{11} + r_{21} + r_{1M} + r_{2M} \leq 1$. In other words, $r_{1M} \leq 0.2$ and $r_{1M} + r_{2M} \leq 0.4$. To show that these inequalities (along with non-negativity constraints) are sufficient, we need to show that the corner points of the polytope described by them can be achieved by dynamic splitting. The rest of the region can then be achieved by time-sharing. The corner points are: $(0,0)$, $(0.2,0)$, $(0.2,0.2)$ and $(0,0.4)$. Of these, the first three can be achieved by one of the static splitting schemes, and therefore by dynamic splitting as well. We only need to show that $(0,0.4)$ is achievable. It can be verified that the following schedule achieves this point: serve the multicast flow with no-splitting for a 0.2 fraction of time and using copy strategy for the remaining time.

This example brings to light an interesting phenomenon. If the unicast demands are unequal at the two inputs (as is the case here), then *we can do better by copying* at the input which has less unicast traffic, than if we did not copy at all. This is surprising at first; copying cells at the input is usually considered wasteful, since it needs more time slots than to directly multicast them. However, copying also gives flexibility in the schedule. The constraint that all cells of a flow must be sent together, is now absent. Hence we have a tradeoff – there is no single winner between copying and no-splitting.

In static splitting, there are several ways to split the flows. The example shows that, **among the different ways to do static splitting, there is no single dominant strategy, in terms of the rate region.** Even if a point is within the rate region of more than one strategy, the delay performance is still different. The strategy in which the required rates are closer to the edge of the rate region will, in general, lead to larger delay. So, the exact way to do the static splitting must be chosen with care. Note that dynamic splitting, which subsumes all other strategies, has the largest rate region, as expected.

C. Earlier Work

Initial work on the problem of scheduling multicast in an IQ switch was based on the copy strategy [14] – multicast cells were replicated at the inputs and treated as unicast cells.

However, this approach reduces the bandwidth available to other traffic in the switch. Besides, a memory speedup is needed.

To circumvent these issues, the idea of using the intrinsic multicast capability was investigated. This idea has implications not only within the switch, but also at the network level. We will discuss the implications to optical networking in Section I-E.

Prabhakar *et al.* [15] considered an IQ switch with intrinsic multicast capability and one FIFO queue per input. The work proposed a scheduling algorithm which allows fanout-splitting and tries to concentrate the residue on as few inputs as possible.

Andrews *et al.* [16] proved that computing the minimum delay multicast schedule is *NP*-hard with and without fanout splitting. The authors also suggested the integration of unicast and multicast traffic in a switch, *i.e.*, allowing unicasts to be served at those inputs and outputs which the multicast schedule leaves idle. This idea was also suggested in [5].

An OQ switch can sustain all admissible multicast traffic. Interestingly, Marsan *et al.* [17] proved that, unlike in the unicast case, the rate region of IQ switches for multicast is strictly smaller than the rate region of OQ switches even with dynamic splitting. This means, not all admissible traffic patterns are sustainable in an IQ switch. The minimum speedup necessary to obtain 100% throughput for all admissible traffic, was shown to grow with the switch size. The paper also defined the optimal online multicast scheduling algorithm in terms of a linear program with a large number of constraints, and showed that the achievable rate region is the convex hull of modified departure vectors.

In [19], Chang *et al.* proposed the load balanced BVN switch and showed that with two crossbars, it is possible to achieve 100% throughput for all admissible multicast traffic. The first unbuffered crossbar performs load balancing and makes the input traffic to the second stage uniform. The second stage is an input-queued Birkhoff-von Neumann switch that runs a sequence of periodic connection states.

In our paper, we use the TWIN approach and view the entire optical network as a single crossbar switch. In this case, it is not possible to use the load-balanced BVN solution directly because for that we would need two crossbars with buffering in between. If we view the network as two switches, then buffering between the switches will have to be implemented using optical buffers, which is a difficult proposition. Alternately, it would require conversion into the electronic domain inside the core of the network, which goes against the argument for having a simple circuit switched all-optical core with buffering only at the edges. In summary, in this paper, we have considered a different problem, where for practical reasons, using two crossbars with buffering in between is a difficult proposition. Hence, we consider a single crossbar switch that uses an offline virtual circuit switching approach.

Another related work is [35]. Here, Caramanis *et al.* used a graph theoretic conflict graph based approach similar to the one we have used, for the problem of unicast switch scheduling in a Banyan network.

D. Contributions of This Work

In this work, we address the problem of computing an offline multicast schedule in an IQ switch, given the average rates of the multicast flows, in a manner akin to BVN. We consider a non-blocking switch such as the crossbar switch in Figure 1. All earlier work focuses on the dynamic splitting case, which results in complicated queue management. With static splitting however, the number of VOQs at each input is drastically reduced, as explained in Section I-A. Besides, static splitting could help reduce the number of reconfigurations of the switch, which is desirable if we use an optical switching fabric. We focus on the no-splitting strategy, but our results can be generalized to the static splitting case by applying them to the split flows.

In summary, the **main contributions** of this work are to provide answers and insight on the following questions:

- 1) *Can we use knowledge of the traffic rate requirements and apply a BVN-like decomposition approach to compute an offline schedule for multicast flows?* We propose a graph theoretic formulation of this problem, using the idea of a conflict graph (see Section III), and show that computing the offline schedule, with no fanout splitting, reduces to the fractional weighted coloring problem on a graph. This leads to a polynomial time algorithm when the graph is perfect.
- 2) *Given the rates of various flows, how can we decide whether they are within the achievable rate region?* We characterize the rate region for the no-splitting strategy in terms of the stable set polytope of the conflict graph, and show that deciding achievability is *NP*-hard. We propose a scheme to decide achievability and compute the offline schedule in polynomial time, when the number of multicasts in the switch is restricted.
- 3) *How does fanout-splitting affect the rate region?* We introduce the concept of static vs. dynamic splitting (see Section I-A). Our rate region results for the no-splitting case readily extend to the static splitting case, when applied to the split flows. We show that the rate region with dynamic splitting is the convex hull of the rate regions of all possible static splitting strategies. We already saw in Section I-B that among the static splitting strategies, there is no single winner in terms of rate region.
- 4) *Can we design online algorithms with low complexity to perform multicast scheduling?* This question is especially important when we do not have knowledge of the average rates of flows in the switch and when issues of speed preclude even moderate polynomial time algorithms, such as bipartite matching or related algorithms. We provide an online heuristic algorithm based on our conflict graph formulation and the *i*-SLIP unicast scheduling algorithm of [4].

E. Implications for Optical Networking

The contributions of this work have implications for optical networks at two different levels – at the switch level and at the network level.

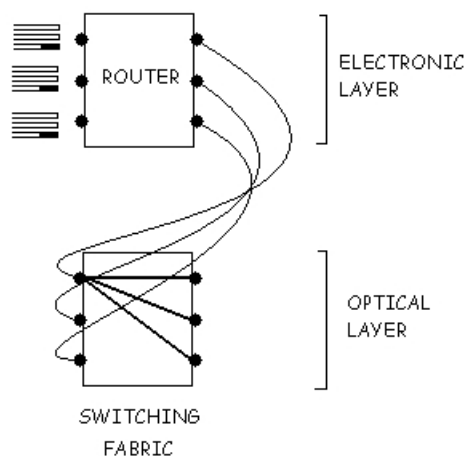


Fig. 4. How should a router exploit the optical switching fabric?

1) Impact at the Switch Level – Optics inside Routers:

High speed routers operate at rates in the order of several tens of gigabits per second. One approach being studied today is whether the use of optics inside the router will help increase these speeds even further [28]. There are two options here. One is the all-optical switch, with fiber delay-line buffers used to buffer packets. The paper by McKeown [28] suggests that, for high-speed Gb/s internet routers that do packet switching, the all-optical approach is probably not feasible any time soon, because of a large buffering requirement, and complicated architecture, involving millions of gates.

The other option is to use electronic buffers with an optical switching fabric. This would require conversion between the electrical and optical domains. With the present state of technology, electronic and optical domains have their own advantages and limitations. Buffering and computation are best done in the electronic domain using semiconductor circuits. Optical buffers and gates currently are onerous and may exhibit limitations. However, electronics also lead to high power consumption and capacity constraints, as compared to optics. In fact, the use of an optical switching fabric will mean huge capacities and very low power consumption in the switch (see Chapter 12 in [29] for a discussion). Another advantage of optical switching fabric is the intrinsic multicast capability. It is possible to send a beam of light to multiple outputs simultaneously using an optical device called the power splitter [32]. An attractive solution is to exploit the strength of the two technologies. This may mean, do the buffering in the electronic domain, and use an optical switching fabric. Then, the intrinsic multicast capability could actually simplify queue management at the electronic layer. The overall idea is to make use of the capabilities of optics to improve the switch throughput without complicating queue management in the electronic domain. Our work assists in this endeavor by taking into account the intrinsic multicast capability while designing the scheduling algorithms.

2) Impact at the Network Level: In order to implement multicast in an optical network, the network layer multicast protocol running in the routers has to create a multicast tree. To realize this physically, the router has one of two options, as

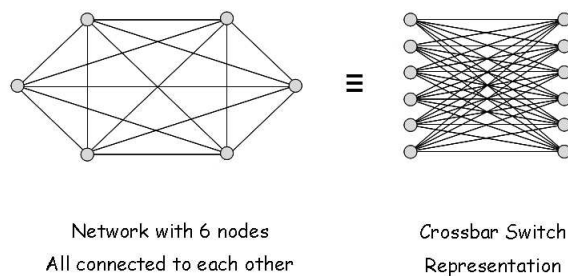


Fig. 5. Viewing the entire network as a crossbar switch

explained in [30] and [31] – it can implement the multicasting in the physical layer or in the network layer.

In the physical layer, there are two approaches. The first approach, known as the *transparent* approach, involves directly using the physical (optical) layer for multicasting using all optical cross-connects. Advantages of the concept of light trees in all-optical networks are discussed in [33]. Note that this requires power splitters along with amplifiers to compensate for the resulting losses. Reference [34] addresses the question of which nodes in the network should be made multicast capable in an all-optical network. The all-optical approach is probably best suited for a circuit switching application. The other approach, called the *opaque* approach, is to convert light to the electronic domain, switch using a crossbar switching fabric and finally revert to the optical domain. Here too, the intrinsic multicast capability of the crossbar fabric can be utilized.

To implement the multicasting in the network layer, we may use IP layer multicasting with physical layer unicasting, which avoids the use of the splitting ability of switches. Here the router replicates the packets and unicasts different copies to the destinations.

At first, it may seem that it is always better to multicast directly in the physical layer, rather than make multiple copies and then unicast them. However, in this work we show that if dynamic splitting is not allowed in the switch, then there is no clear winner between the copy strategy and the direct multicast strategy in terms of the rate region. This fact is clearly seen from the simple 2×2 switch example given in Section I-B. This effect occurs in each switch and hence, in a network of switches, the effect will accumulate. The decision of whether to split the flow into unicasts, and if so, whether to do it at the ingress of the network or further downstream, is thus not a trivial one, and deserves careful analysis.

One example of the transparent approach at the physical layer is the *time-domain wavelength interleaved network* or TWIN, which was proposed in [7] and [8]. The main idea in TWIN is to have a separate wavelength for each destination, and a simple core consisting of wavelength-selective switches with preset configurations that route the light streams to the correct destinations. In this case, all the complexity is shifted to the edge of the network, where ultrafast tunable lasers are used in the transmitters to time-share among the flows. The switches within the network are thus simplified, without the need for optical buffers or fast reconfiguration. Since each transmitter can transmit only one wavelength at a time, and

each receiver can receive only one wavelength at a time, careful scheduling of the transmissions is required.

The ideas of our work can be applied to networks like TWIN, to extend them to multicast flows, *i.e.*, point to multipoint flows. In [8], the scheduling problem is formulated as an extension to the maximum matching problem in a bipartite graph. In the case of zero propagation delay, the problem is identical to the crossbar switch scheduling problem. In our work too, the entire network may be viewed as a crossbar switch (see Figure 5). However, in addition to bipartite matchings, we also allow switch configurations where flows may be directly multicast to several destinations. Inherently, optical networks are suited to such direct multicast using power splitters. In particular, the static fanout splitting of a flow that we consider in this work, is directly applicable to a TWIN-like setting, where the flow patterns are decided beforehand and are kept fixed.

The rest of the paper is organized as follows. Section II describes the BVN switch. Section III presents the conflict graph formulation and the main result about the rate region for multicast in an IQ switch. Section IV proves that deciding achievability of a given set of rates is *NP*-hard with and without fanout splitting. Offline schedule computation is shown to be reducible to fractional weighted graph coloring, which gives a polynomial algorithm when the conflict graph is perfect. When the conflict graph is not perfect, a polynomial time scheme is proposed to decide achievability and compute the offline schedule when the number of multicast flows in the switch is moderate. Section V contains an online heuristic scheduling algorithm for no-splitting that combines the ideas of *i*-SLIP and our conflict graph approach. We also present simulation results comparing the no-splitting strategy with the copy strategy. Finally, section VI gives the conclusions. Appendix lists the definitions of all graph theoretic terms used in this paper. The proofs of some of the theorems have been moved to Appendix for clarity of presentation.

II. THE BIRKHOFF-VON NEUMANN SWITCH

The Birkhoff-von Neumann (BVN) switching algorithm [9] provides rate guarantees as well as cell delay guarantees for IQ crossbar switches, while achieving 100% throughput. The gist of the BVN approach is to decompose a set of demands into a convex combination of permutation matrices, which naturally correspond to switch configurations.

Consider an $N \times N$ IQ crossbar switch. Let $R = (r_{ij})$ be the required rate matrix, where r_{ij} represents the rate from input i to output j . The BVN theorem ([2], [3]) states that any $N \times N$ doubly stochastic matrix can be expressed as a convex combination of no more than $(N^2 - 2N + 2)$ permutation matrices. Applying this theorem to the rates, we conclude that, if:

$$\sum_{i=1}^N r_{ij} \leq 1, \forall j = 1, \dots, N \text{ and } \sum_{j=1}^N r_{ij} \leq 1, \forall i = 1, \dots, N \quad (5)$$

(*i.e.*, the rate matrix is doubly sub-stochastic), then, there exist positive numbers ϕ_k and permutation matrices P_k ($k =$

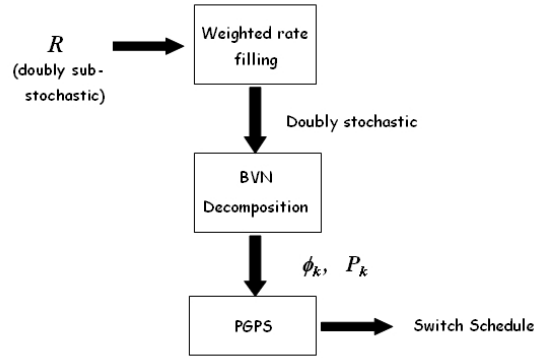


Fig. 6. The BVN switching algorithm

$1, \dots, K$, where $K \leq N^2 - 2N + 2$) such that,

$$R \leq \sum_{k=1}^K \phi_k P_k, \quad \sum_{k=1}^K \phi_k = 1.$$

The BVN switching algorithm was proposed in [9], [10], [11] and is shown in Figure 6. The first step is to replace the possibly sub-stochastic rate matrix by a doubly stochastic matrix which majorizes it. This step is quite simple and the algorithm is given in [9]. An alternate, less complex, algorithm is also given in [1], called the *weighted rate filling algorithm*. The matrix is then decomposed into permutation matrices. It can be shown that this problem of expressing a doubly stochastic matrix as a convex combination of permutation matrices, is equivalent to the problem of finding perfect matchings in a bipartite graph. Note that, the matching problem is solved offline with a complexity of $O(N^{2.5})$, where N is the switch size. Having obtained the decomposition into permutation matrices offline, the scheduling itself may be done online, according for instance, to an algorithm in [9] based on Packetized Generalized Processor Sharing (PGPS) [12], [13]. This algorithm approximates the strategy of keeping the switch in the state given by P_k for a fraction of time ϕ_k , without the granularity problems associated with framing. Another way to implement BVN is to randomly pick, in each time slot, one of the P_k 's with a probability ϕ_k . One can show that this randomized strategy will lead to stability in the long run.

III. THE MULTICAST RATE REGION

The aim is to find a BVN-like decomposition of the rates into a convex combination of switch states, for the multicast case, and to identify the set of rates for which such a decomposition is possible. Before we get into that problem, let us look at an interpretation of the unicast rate region using a graph theoretic model.

A. Interpreting the Unicast Rate Region

If we view the switch as a complete bipartite graph with the rates as the edge weights, then the unicast capacity region can be interpreted in terms of the matching polytope⁶ of the graph. This is because any point inside the polytope can be expressed

⁶All graph theoretic terms have been defined in the Appendix.

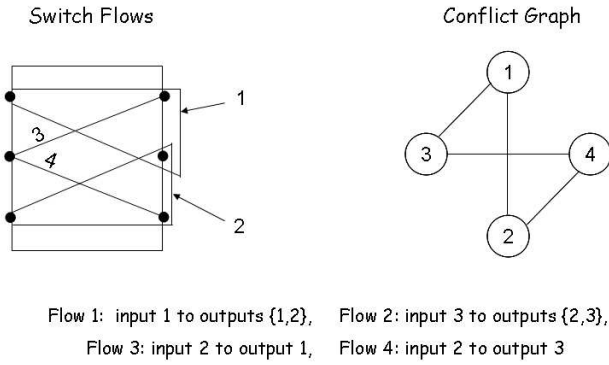


Fig. 7. An example of a traffic pattern and its conflict graph

as a convex combination of matchings, which immediately gives a schedule. The matching polytope of the complete bipartite graph has been characterized completely ([20]). The Birkhoff-von Neumann decomposition into permutation matrices is simply the matching decomposition of the bipartite graph. A similar approach that uses edge coloring of bipartite graphs to find a schedule for unicast traffic in Clos network switches is discussed in [26].

If we try to extend this model to the multicast case, it becomes cumbersome, because the multicast flows will have to be represented by hyperedges. However, there is another way to represent the traffic pattern, which does not necessitate the use of hypergraphs. This is the conflict graph representation, which we discuss next.

B. Conflict Graph Formulation

Definition 3.1 (Flow): A flow is a 2-tuple consisting of an input and a subset of outputs corresponding to the destination set of the multicast.

Definition 3.2 (Conflict graph): The conflict graph for a given traffic pattern is defined as a graph $G = (V, E)$ where:

$V =$ set of all flows to be served

$E = \{(v_i, v_j) \mid \text{flows } i, j \text{ cannot Coexist in a valid switch configuration}\}$

There is one vertex corresponding to each flow. An edge connects two vertices if the corresponding flows cannot co-exist in any valid configuration of the switch. (Note that two flows cannot co-exist if they share an input or an output.) Such flows are said to “conflict” with each other. The conflict graph models the connection constraints in the switch. Figure 7 shows an example of a traffic pattern and its conflict graph.

A valid switch configuration consists of a set of flows that can co-exist. In the conflict graph, this maps to a set of vertices no two of which are connected - in other words a *stable set*. For a given set of unicast and multicast flows, the achievable rate region is thus the *stable set polytope* of the conflict graph of the flows, because there is a one-to-one correspondence between a time-sharing of valid switch states and a convex combination of stable sets of the conflict graph. A convex combination of stable sets gives a schedule where the flows in a particular stable set are served for a fraction of time equal to the coefficient in the combination.

C. The Main Result

The discussion above immediately leads to the following theorem:

Theorem 1: *The capacity region for the multicast case with no fanout splitting is the stable set polytope of the conflict graph.*

This theorem can be extended to static splitting by using the conflict graph of the split flows. The rate region is then obtained by projecting the polytope, such that, all split flows corresponding to the same original flow get the same rate.

Note that, while our formulation does not directly tell us what is the optimal way to split the flows, it allows us to compute the achievable rate region given a particular static splitting strategy, and hence it allows to compare two different strategies. From the example in Section I-B, it is clear that there is no single dominant strategy. Computing the optimal choice is still an open problem, and constitutes future work. It is expected that the optimal choice will critically depend on the traffic pattern and the rates.

A result similar to the theorem above was shown by [36]. In that paper, the authors consider the problem of online scheduling of a system of queues, with constraints on which queues may be activated simultaneously. A set of queues that can be served simultaneously is called an *activation vector*. Knowledge of the queue sizes is used to find which set of queues should be activated in each time slot. A maximum weighted activation vector policy is shown to achieve all rates within rate region, which in turn, is shown to be the convex hull of all possible activation vectors. The multicast switch setup that we consider also consists of a set of queues with pairwise constraints. A valid activation vector corresponds to a set of flows which do not conflict amongst each other, *i.e.*, a set of flows that form a stable set in the conflict graph.

However, our work differs from [36] because we consider a different problem – the problem of offline scheduling. We do not assume knowledge of queue occupancies. Instead, we use knowledge of average arrival rates of the flows to compute an offline schedule in terms of the conflict graph. In Section IV, we show that computation of an offline schedule is equivalent to fractional weighted coloring of the conflict graph, where the vertex weights are chosen to be the rates of the flows.

D. When is the Conflict Graph Perfect?

In general, the characterization of the stable set polytope of a graph is not completely known. The rate region can be explicitly described only in some special cases. One important case is when the conflict graph is *perfect*. In this case, the rate region is completely characterized by non-negativity conditions and the clique inequalities:

$$\sum_{v \in Q} x_v \leq 1 \text{ for every clique } Q \text{ in the conflict graph.}$$

where x_v denotes the rate of the flow corresponding to vertex v . In general, the clique conditions are necessary but not sufficient. Some examples of traffic patterns that lead to perfect conflict graphs are discussed below.

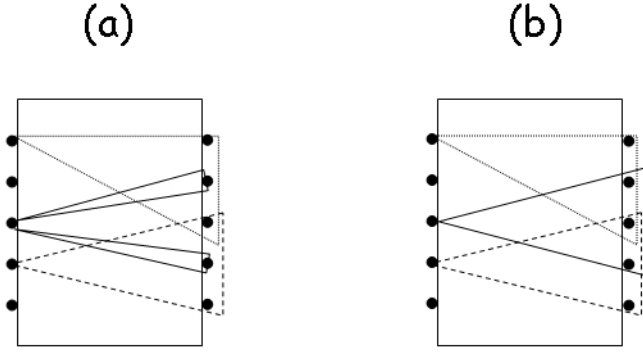


Fig. 8. Example: The multicast from input 3 does not form a contiguous interval in (a), but can be completed to one, without any new conflicts as in (b). The conflict graph is thus an interval graph and is perfect.

1) *Unicast*: The unicast case gives rise to a conflict graph which is the *line graph of a bipartite graph*. The line graph of a bipartite graph is a well-known example of a perfect graph (see [20]). Hence, the clique inequalities completely characterize the stable set polytope. The cliques in the conflict graph of a unicast traffic pattern correspond to a set of flows that begin at the same input or flows that terminate at the same output. The condition derived from the BVN theorem that the rate matrix should be doubly sub-stochastic, thus gives the same rate region as the stable set polytope of the conflict graph. The conflict graph approach thus reduces to the BVN approach in case of unicast.

2) $2 \times N$ Switch with Arbitrary Traffic:

Theorem 2: The conflict graph of a $2 \times N$ switch with an arbitrary traffic pattern (unicast and multicast), with no fanout splitting, is perfect.

Proof: Consider the complement of the conflict graph. In this graph, two vertices will have no edge between them if and only if they cannot be simultaneously served in the switch. The vertices of this graph can thus be partitioned into 2 stable sets, according to which input the corresponding flow originates from, since two flows from the same input can never be served simultaneously. This proves that the conflict graph of any traffic pattern in a $2 \times N$ switch with no fanout splitting is the complement of a bipartite graph, which is known to be perfect [20]. ■

Combining with Theorem 1 we get the following corollary.

Corollary 1: The rate region of a $2 \times N$ switch for an arbitrary traffic pattern (including unicast and multicast) with no fanout splitting is given by the non-negativity and clique inequalities of the conflict graph of the pattern.

3) *The Interval Graph Case*: Suppose unicasts are absent, and there is one multicast flow per input. If for each multicast, all the outputs are successive outputs (*i.e.*, one contiguous set), then the conflict graph is an interval graph, which is perfect. Note that, if permuting the outputs leads to contiguous fanouts, then also, the graph is perfect. Finding such a permutation is a well-studied problem known as the *consecutive arrangement problem*. One can also convert a given traffic pattern into the interval graph case by completing non-contiguous fanouts as shown in Figure 8.

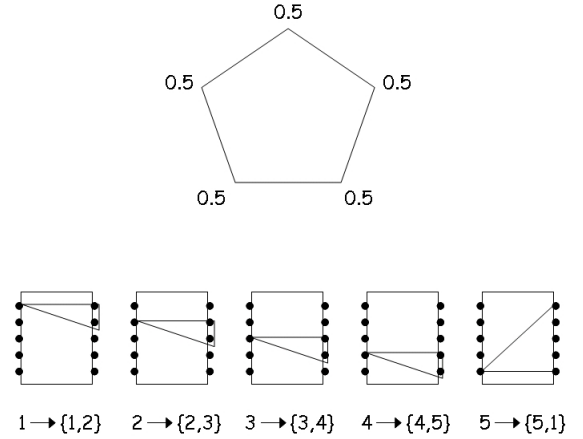


Fig. 9. Example of case where clique inequalities do not suffice to characterize the rate region

4) *Mixture of Unicast and Broadcast*: In this example, every multicast is a broadcast, that is, it has to be delivered to all outputs. The unicast flows are present, in addition to these broadcasts. Thus the conflict graph can be viewed to consist of two parts - a unicast part which is basically the line graph of a bipartite graph L , and a multicast part M , which happens to be a clique in this case, as any broadcast flow will conflict with any other flow. Every vertex in L is connected to every vertex in M . One can show easily that this graph is perfect.

5) *A Non-Perfect Case*: Figure 9 shows an example when the graph is not a perfect graph and the resulting stable set polytope cannot be characterized completely by the clique inequalities. This is the simplest example of a non-perfect graph. Note that this traffic pattern may not occur naturally.

More generally, if both unicast and multicast flows are present, then the conflict graph may not be perfect, except when all the multicast flows are broadcast. This is because a multicast may form an *odd hole* in the conflict graph, along with flanking unicasts. This directly breaks the perfect nature of the graph, because of the strong perfect graph theorem [20]. One way to address this problem is to modify the traffic pattern such that its conflict graph becomes perfect at the cost of loss of optimality.

IV. DECIDING ACHIEVABILITY AND COMPUTING THE OFFLINE SCHEDULE

We prove hardness results for deciding whether a given set of rates is within the rate region.

Lemma 1: Given a general graph G , there exists a multicast traffic pattern in a $|V(G)| \times (|E(G)| + N_{isolated})$ switch, with G as the conflict graph corresponding to the no-splitting case, where $N_{isolated}$ is the number of isolated vertices of G .

Proof: It is known that every graph is the intersection graph of some family of sets. For example, for each non-isolated vertex $v_i \in V(G)$, define the set S_i to be the set of edges intersecting v_i . If v_i is an isolated vertex, then S_i is a singleton set with a new element. Then G is the intersection graph of the family of sets $\{S_1, S_2, \dots, S_{|V(G)|}\}$. Now, create a traffic pattern in a $|V(G)| \times (|E(G)| + N_{isolated})$ switch such that, there is a multicast flow from input i , with fanout being

the set S_i . The conflict graph of this flow pattern is precisely G , since there are no conflicts at the inputs, and since G is the intersection graph of the fanout sets. ■

Theorem 3: The problem of deciding whether a given rate requirement vector is achievable using the no-splitting strategy is NP-hard.

Proof: From Lemma 1, given any graph G , there exists a traffic pattern such that G is the conflict graph for that pattern. Therefore, the problem of characterizing the rate region for a general traffic pattern is equivalent to the problem of characterizing the stable set polytope of a general graph.

The problem of finding the maximum weighted stable set can be viewed as the maximization of a linear function over the stable set polytope. This implies that the problem mentioned in this theorem is precisely the membership problem corresponding to the maximum weighted stable set optimization problem. If there is an oracle that efficiently decides whether a given point is within a convex polytope or not, then there is an efficient way to optimize over that polytope, since the membership problem is at least as hard as the weak optimization problem [22].

Thus, we essentially have a reduction from the maximum weighted stable set problem to the problem of deciding the achievability of a given rate vector. The maximum weighted stable set problem is known to be NP-hard [23]. This proves the theorem. ■

Next, we show that the hardness result can be extended to the case when dynamic splitting is allowed. For details of the proof, refer to Appendix .

Theorem 4: The problem of deciding whether a given rate vector is within the rate region of the no-splitting case can be reduced to an instance of the corresponding problem in the fanout-splitting case.

Corollary 2: The problem of deciding achievability in a multicast switch when fanout-splitting is allowed, is NP-hard.

A. Offline Schedule with Perfect Conflict Graphs

We address the problem of computing the offline schedule by decomposing the rate requirement, in a manner similar to the BVN algorithm. This approach gives a polynomial time algorithm for the case of perfect conflict graphs.

Theorem 5: The problem of computing the decomposition of the given rate requirements into switch connection states reduces to the problem of fractional weighted graph coloring⁷.

Proof: Consider the conflict graph. Assign to each vertex, a weight equal to the rate requirement for the corresponding flow (normalized to the line rates). Suppose there is an algorithm to obtain the minimum fractional weighted coloring for this graph, then such a coloring immediately gives an offline schedule for the switch: each color (stable set) corresponds to a valid switch connection state, and the weight of the color in the coloring gives the fraction of time for that connection state. Since the coloring is valid, each flow will receive at least as large a fraction of time, as its required rate. If the fractional weighted chromatic number (*i.e.*, the sum of weights of all colors) is more than 1, that means the given rate vector is

outside the rate region, and hence no schedule can be found. ■

Grötschel *et al.* [21] proved that the fractional weighted coloring problem is NP-hard for general graphs, but can be solved in polynomial time for perfect graphs.

We saw in Section III-D, several examples of traffic patterns which give rise to perfect conflict graphs. For instance, the unicast case leads to a perfect conflict graph, and the coloring problem is therefore easy. In fact, weighted coloring of the conflict graph is equivalent to edge coloring of a bipartite multigraph. This approach to unicast scheduling has been explored in [25].

B. The $K \times N$ Switch with Arbitrary Traffic

We now consider a $K \times N$ switch with an arbitrary traffic pattern. We show that if K is a constant that does not grow with N , then the offline schedule can be computed in polynomial time with respect to the number of flows in the switch.

Lemma 2: The vertices of the conflict graph of any traffic pattern in a $K \times N$ switch can be partitioned into K sets, each of which induces a clique.

Proof: Group all vertices that correspond to flows from the same input into one set. There are K such sets, one for each input. No vertex (flow in the switch) can be part of more than one such set, since each flow has a unique input. Also, every flow must be in one of the sets. Thus, we have a partition of the vertex set into K cliques. ■

We now use this property of the conflict graph to infer that fractional weighted coloring problem can be solved in polynomial time. Note that, this means that the offline schedule can be computed in polynomial time.

Theorem 6: The problem of computing an offline schedule corresponding to an arbitrary traffic pattern in a $K \times N$ switch can be solved in time that is polynomial in the number of flows in the switch, provided K is a constant with respect to N .

Proof: It is known in graph theory literature that the maximum weighted stable set problem can be solved in time polynomial in the number of vertices for a (p, q) -colorable graph (where p and q are constants), provided $q \leq 2$ [37].

From Lemma 2, the vertex set of the conflict graph for an arbitrary traffic pattern can be partitioned into K cliques. Hence, the conflict graph is a $(K, 0)$ -colorable graph, where K is a constant with respect to N . Hence, the maximum weighted stable set problem can be solved in time polynomial in the number of flows in the switch.

Now, the results in [21] imply that for any collection \mathcal{G} of graphs, if the weighted stable set problem for graphs in \mathcal{G} is polynomial-time solvable for any weight function, then the fractional weighted coloring problem for graphs in \mathcal{G} can also be solved in polynomial time.

From Theorem 5, a solution of the fractional weighted coloring problem in the conflict graph gives an offline schedule. Thus, for a $K \times N$ switch, the offline schedule can be computed in time polynomial in the number of flows in the switch. ■

⁷See the Appendix for the problem statement

C. Polynomial-time Offline Schedule with Moderate Multicast

We now present an algorithm to decide the achievability of a given rate vector, with no fanout-splitting. Here, we do not assume that the conflict graph is perfect. The algorithm runs in time polynomial in the switch size (N) if the number of multicasts is restricted in the following way – if k inputs in the switch receive multicast flows, each receiving at most m multicast flows, then $(m+1)^k$ should be a polynomial function of N . The algorithm naturally gives a schedule to achieve the rates in a stable manner.

Algorithm 1:

INPUT: A rate requirement vector \mathbf{r}_o ; a traffic pattern in an $N \times N$ switch, with all possible unicasts and k multicasting inputs each handling up to m multicast flows.

OUTPUT: Is \mathbf{r}_o within the achievable region corresponding to the traffic pattern, in the no-splitting case? If yes, give a schedule to sustain the rates.

- 1) Let A be any subset of the multicast flows that can be simultaneously served. Let R_A be the rate region for the traffic pattern with the condition that the multicasts $\{M_i | i \in A\}$ are all always being served, while the other multicasts are always off. Compute R_A for all possible conflict-free subsets A of the multicast flows.
- 2) Verify whether \mathbf{r}_o lies in the convex hull of all the R_A 's. The answer to this question is the output.

Let $\mathbf{B}_j \mathbf{x} \leq \mathbf{c}_j, j = 1, 2, \dots, J$ be the set of convex regions representing the R_A 's. Verifying whether a point \mathbf{r} is in the convex hull of these regions is equivalent to verifying whether this linear program in the variables \mathbf{y}_j and ϕ_j is feasible:

$$\sum_{j=1}^J \phi_j = 1; \quad \phi_j \geq 0; \quad \mathbf{r} = \sum_{j=1}^J \mathbf{y}_j; \quad \mathbf{B}_j \mathbf{y}_j \leq \phi_j \mathbf{c}_j \quad \forall j = 1 \text{ to } J. \quad (6)$$

The proofs of the following lemmas and theorems are given in Appendix .

Lemma 3: Algorithm 1 is correct, *i.e.*, the rate region for the given traffic pattern is precisely the convex hull of the regions R_A for all possible conflict-free subsets A of the multicast flows.

Lemma 4: Suppose that fanout-splitting is not allowed. The rate region for a given traffic pattern with the condition that all the multicast flows are simultaneously served throughout the schedule, is given by:

Case 1: The rate region is empty if any two of the multicasts overlap.

Case 2: If no two of the multicasts overlap, then the region is:

$$\begin{aligned} \sum_j r_{ij} &\leq 1, \quad \forall \text{ non-multicast inputs } i \\ \sum_i r_{ij} &\leq 1, \quad \forall \text{ outputs } j \text{ outside any multicast's fanout} \\ r_{ij} &= 0, \text{ if } i \text{ is a multicasting input or } j \text{ is within} \\ &\quad \text{some multicast's fanout} \\ r_{M_i} &= 1, \text{ for all multicasts } M_i. \end{aligned}$$

where r_{ij} is the unicast rate from input i to output j .

Corollary 3: For each subset A_i , there is an explicit characterization of R_{A_i} using polynomial number of inequalities with respect to N .

Proof: From Lemma 4, the rate region can be explicitly specified using at most one inequality for each input, output and multicast flow. Since the number of multicast flows ($\leq km$) is polynomial in N , the number of inequalities is polynomial in N . ■

Theorem 7: Consider a traffic pattern in an $N \times N$ switch, with k multicast inputs each with m multicasts and any number of unicasts. Suppose that fanout-splitting is not allowed. Then Algorithm 1 is a scheme to decide whether a given rate requirement vector is achievable or not, in time polynomial in N if $(m+1)^k$ is a polynomial function of N . In this sense, deciding achievability is fixed parameter tractable in the number of multicasts.

Proof: By Lemma 3, Algorithm 1 is correct. We now have to show the complexity result. The number of R_A 's is not more than $(m+1)^k$, since in any conflict-free subset of multicasts, at most one multicast flow can be picked from each multicasting input. By Corollary 3, for each subset A_i , there is an explicit characterization of R_{A_i} using polynomial number of inequalities with respect to N . Verifying whether the given rate requirement vector is in the convex hull of the R_{A_i} 's is equivalent to verifying feasibility of the LP given in equation (6). Verifying feasibility of an LP can be done in time polynomial in the number of constraints and variables, using the ellipsoid algorithm [24]. If $(m+1)^k$ is a polynomial function of N , then the number of inequalities in (6) is polynomial in N and the theorem follows. Hence, the theorem follows. ■

V. HEURISTIC ONLINE ALGORITHMS

In the situation that the average rates of the flows are not available, we need online algorithms with a low complexity to perform the scheduling. Note that, even moderate polynomial scheduling algorithms like the bipartite matching algorithm (for unicast) cannot be implemented in an online setting in high-speed switches. *i*-SLIP is a well known online unicast scheduling algorithm proposed by McKeown in [4], to address these issues. We propose an online algorithm for multicast, which uses the conflict graph idea to extend the *i*-SLIP unicast algorithm for multicast. This is for the case when fanout-splitting is not allowed.

We compare it with the ESLIP algorithm, which was suggested in [5] as an extension of *i*-SLIP for multicast. We use a modified version of ESLIP for the comparison so as to allow multiple VOQs for multicast flows. This is for the case when dynamic splitting is allowed.

We also include in the comparison the copying strategy, where the fanout is split completely and the split flows are treated as unicasts. They are then scheduled using *i*-SLIP.

Using simulations, we show that there is no clear winner between our online (clique) algorithm for no-splitting and the *i*-SLIP based copying strategy. The modified ESLIP always performs better than the other two. This is expected since dynamic splitting subsumes the other splitting strategies. However, dynamic splitting has its own difficulties with respect to implementation, as explained in Section I-A.

A. Clique Algorithm

The motivation for this algorithm is the interpretation of i -SLIP in terms of the conflict graph formulation (Section III-B). The i -SLIP algorithm is an iterative algorithm, with each iteration involving three steps, as described in [4]. The key idea in our proposed algorithm is to transfer the job of selecting grants from the outputs to cliques in an output conflict graph⁸. Similarly, the job of accepting grants is now done by each clique in the input conflict graph⁸. The grants and accepts are not for inputs or outputs, but for flows. Just as i -SLIP computes a maximal matching in the switch iteratively, the clique algorithm computes a maximal stable set in the conflict graph.

In this algorithm, *input conflict graph* is the conflict graph obtained by considering conflicts between flows only on the input side. Similarly, *output conflict graph* is the conflict graph obtained by considering conflicts between flows only on the output side. The algorithm is as follows:

Algorithm 2: Clique Algorithm:

INPUT: A traffic pattern in an $N \times N$ switch, with multicasts and unicasts, and the current state of the VOQs (empty or occupied?).

OUTPUT: The switch connection state for the current time slot.

- 1) Form the input and output conflict graphs and remove those vertices (flows) for which there are no packets at the head of the corresponding queue.
- 2) Find all maximal cliques in both the input conflict graph and the output conflict graph
- 3) GRANT: For all $i=1$ to (number of maximal output cliques)
 - a) Every output clique maintains a round robin pointer. Output clique i chooses that vertex (flow) from that clique, which appears next in a round robin schedule based on the pointer, as in i -SLIP.
 - b) The neighbors of the chosen flow are removed from the output conflict graph, and the cliques are correspondingly updated.
- 4) ACCEPT: For every input clique
 - a) Among all the grants that belong to this input clique, that flow is accepted which appears next on a round robin schedule, based on a pointer.

The above is one iteration of the algorithm. Successive iterations proceed in a similar manner with flows which have not yet been accepted. As in i -SLIP, pointers are updated only at the end of the first iteration. The input conflict graph consists of disjoint cliques. But this is not true of the output side. This is why, the loop in step 2 requires that the neighbors of the chosen vertex be removed from the output conflict graph. This automatically implies that the order in which the output cliques are considered will affect how the algorithm performs. This asymmetry is avoided by allowing the loop to start with a different clique in different cell slots, in a round robin manner.

⁸The input (output) conflict graph is a conflict graph whose edges correspond to conflicts only on the input (output) side.

Note that this algorithm does not split the fanout at all. Finding all maximal cliques in the conflict graph can be done in polynomial time when the conflict graph is perfect. Even otherwise, it is important to note that, the cliques need to be found only once, offline. Subsequently, in a given cell slot, the vertices corresponding to the flows with empty queues are simply removed from the conflict graph. The cliques in the resulting graph can be found easily by just removing the corresponding vertices from the cliques in the original conflict graph.

In every time slot, information about the current state of the VOQs (*i.e.*, whether they are empty or occupied) needs to be exchanged. As a result, SLIP-like algorithms may have problems of communication overhead. However, if we choose to change the switch configuration not after every time slot, but instead over a frame of slots, then this overhead can be amortized over several time slots. Thus, while the solution is still online, it works over a large enough burst of packets at a time, so that the communication overhead is not dominant any more.

B. A modification of ESLIP

In this algorithm, we allow fanout-splitting as in ESLIP. The main idea is to get as many packets across as possible within a time slot in a greedy manner. The algorithm is closely related to i -SLIP:

Algorithm 3: Modified ESLIP:

INPUT: The current state of the VOQs (empty or occupied?) in an $N \times N$ switch with unicasts and multicasts.

OUTPUT: The switch connection state for the current time slot.

- 1) REQUEST: Each input sends a request to all outputs for which it has a packet whether unicast or multicast.
- 2) GRANT: This is identical to i -SLIP. The output accepts that input which appears next in the round robin schedule, without consideration of whether it is a unicast or a multicast packet.
- 3) ACCEPT: The list of values the input pointer can take includes one value for each output plus one value for each multicast flow originating at that input. Among the outputs which have given grants, the input accepts that one which appears next in this list. If the next in the list is a multicast flow, the input accepts all granting outputs which lie within the fanout of that flow.

Once the input has accepted an output, it does not participate in future iterations. However, if the input has chosen a multicast flow, then in future iterations, it remains open to grants from other outputs within the fanout of that flow. As in i -SLIP, pointers are updated only in the first iteration, whenever any input successfully accepts any output or set of outputs.

In terms of implementation, this algorithm involves a minor modification of i -SLIP. Whereas ESLIP has a separate pointer that is used commonly for all multicast flows, in our algorithm, there are just extra values in the list of values that the input pointer cycles through, to indicate the multicast flows. This algorithm works in switches where fanout-splitting is allowed.

It is a modification of ESLIP for the case when there are multiple VOQs for multicast flows.

C. Numerical Illustration

The simulation results reported in this section are not intended as an empirical evaluation of the heuristic algorithm presented earlier, in comparison to the existing schemes. Instead, the intention is to illustrate the fact that there is no single dominant static splitting strategy in terms of throughput. In Section I-B, we demonstrated this fact using examples in a 2×2 switch. Now, we use the algorithms proposed above to demonstrate a similar effect in an online setting in a bigger switch.

Each choice of static splitting leads to a different rate region. Consider a rate point that is within the rate region of two different choices of static splitting. Both choices will sustain the rate required, but the delay could be different. Since delay generally goes up as we approach the edge of the rate region, we expect that if we use a strategy in which the rate point is very close to the boundary of the rate region, then we shall incur greater delay than if we use a strategy in which the rate point is well within. To study this effect in greater detail, we have performed simulations of a 16×16 multicast switch.

1) *Simulation Setting — 16×16 switch:* The simulations were carried out using the SIM software [27]. Uniform i.i.d. Bernoulli traffic was used for all flows. The traffic pattern consisted of all possible unicasts and a set of broadcast flows, one from each input going to all outputs. Two cases were simulated.

Case 1: $r_{i,j} = i/320, \forall j; r_{i,B} = 0.005\alpha$ for $i = 1$ to 13 ;

$$r_{14,B} = 0.018\alpha, r_{15,B} = 0.015\alpha, r_{16,B} = 0.012\alpha \quad (7)$$

where $r_{i,j}$ refers to the unicast rate from input i to output j and $r_{i,B}$ refers to the rate of the broadcast flow from input i , and α is a parameter used to control the load.

In the second case, the unicast rates are the same as the first one. The broadcast rates are however different:

Case 2: $r_{1,B} = 0.03\alpha, r_{i,B} = 0.02\alpha$ for $i = 2$ to 7 ,

$$r_{8,B} = 0.01\alpha, r_{i,B} = 0.005\alpha \text{ for } i = 9 \text{ to } 14,$$

$$r_{15,B} = r_{16,B} = 0.003\alpha \quad (8)$$

The load is increased in the following manner. The unicast rates are kept constant, while the broadcast rates are increased in a proportionate manner by varying the parameter α from 0 to 1. The above choice of rates is such that, when $\alpha = 1$, the rate point in case 1 is close to the boundary of the copy strategy rate region, but well within the region of the no-splitting strategy; and in case 2, the situation is reversed.

2) *Inferences:* Figure 10 shows the delay versus multicast fraction plot for these two patterns respectively. Three switching algorithms are compared — the clique algorithm, the modified ESLIP and the i -SLIP. The clique algorithm represents the no-splitting strategy, while the i -SLIP represents the copy strategy⁹. The x-axis of the plot is the average multicast fraction across all the inputs (*i.e.*, the ratio of the

⁹When using i -SLIP for multicast, we assume that a copy network has performed complete fanout splitting of the multicast flows, and has fed the multiple copies into corresponding VOQs.

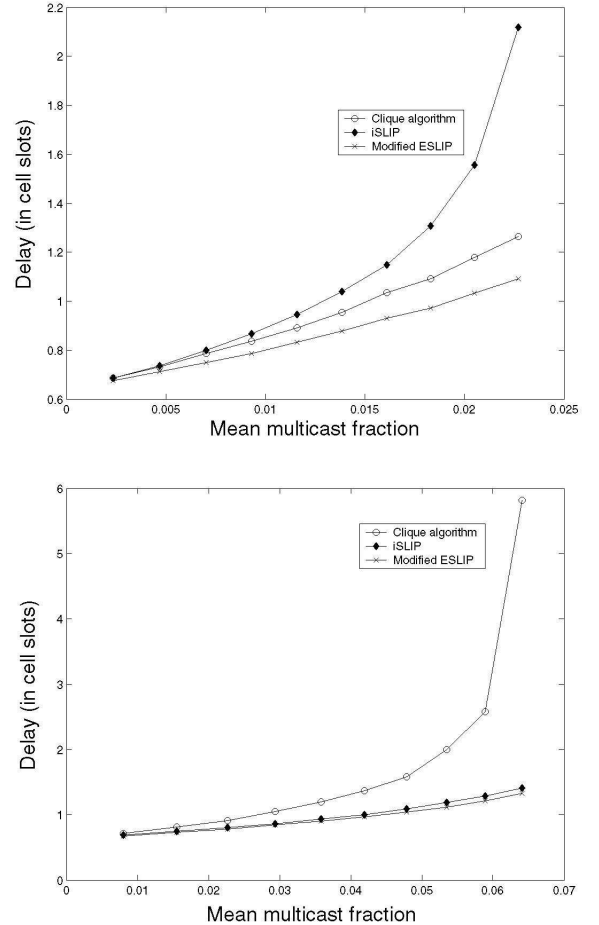


Fig. 10. Delay versus multicast fraction: 16×16 : cases 1 and 2 respectively. Simulation settings are given by Eqn.(7) and (8)

multicast flow rate to the total incoming rate), which is a measure of the load. The y-axis is the delay of the packets averaged over all the flows — unicast and multicast. In all the three algorithms, the number of iterations is set at three.

The first pattern corresponds to a rate point which is near the boundary of the rate region of the copy strategy, but is well within the region of the no-fanout-splitting rate region. This is why the delay is much less for the clique algorithm compared to copy strategy (i -SLIP). The rate point for the second pattern is near the edge of the no-splitting rate region, but is within the region corresponding to complete fanout splitting (*i.e.*, copying). This explains why the i -SLIP does better here in terms of delay, than the clique algorithm.

In both the patterns, modified ESLIP performs better than the other two algorithms. This can be attributed to the fact that modified ESLIP allows dynamic splitting, which subsumes all other strategies, as we saw in Section I-A

The simulation corroborates the earlier statement that among the various static splitting strategies, there is no single dominant strategy. It gives an example where there is no winner between copying and no-splitting, in terms of delay.

VI. CONCLUSIONS

This paper presents a new approach to multicast switching that makes use of prior knowledge of the traffic pattern and

the average rates of the flows, on the lines of the Birkhoff-von Neumann rate-decomposition based switch for unicast. Using a graph-theoretic formulation, we have shown that the rate region for the no-splitting strategy is the stable set polytope of the conflict graph. The result can also be used to obtain the rate region for any particular choice of static splitting, by considering split flows while forming the conflict graph.

We have also proved hardness results for deciding achievability and finding the offline schedule with static splitting. Under a restriction on the number of multicasts, we have provided a polynomial time algorithm for computing the offline schedule. We have shown that the offline schedule computation problem reduces to the fractional weighted graph coloring problem, which implies a polynomial time algorithm for the case of perfect conflict graphs.

We have shown through simulations that there is no single dominant strategy among the various possible static splitting choices, even in an online setting. Dynamic splitting subsumes the other approaches in terms of rate region, but it has implementation difficulties.

The results presented here have implications in the design of optical networks, especially in the context of the TWIN architecture based approach of [7], where the entire network is viewed as a switch with buffering at the edges and circuit switching in the core. The intrinsic multicast capability we have assumed is a naturally available option in the optical domain. Besides, the static splitting case that we study leads to less frequent reconfiguration of the network, and is thus more practical, especially in high-speed optical networks.

An important line of future work is to provide approximation algorithms for computing the offline schedule.

APPENDIX

Graph-Theoretic Definitions:

This appendix provides the definitions of several graph theoretic terms used throughout this paper. These definitions are based on [20]. Consider a graph $G = (V, E)$, where V is the vertex set and E is the edge set.

Definition 1.1 (Perfect matching): A *matching* on G is a set of edges no two of which share a vertex. A *perfect matching*, is a matching that covers all the vertices in the graph.

The incidence vector of a matching M in G , denoted χ^M is a binary vector of length $|E|$, such that, $\chi^M(e) := 1$ if edge $e \in M$; $\chi^M(e) := 0$ if $e \notin M$.

Definition 1.2 (Matching polytope): The *matching polytope* of G is defined as: $M(G) := \text{conv}\{\chi^M \in \mathbb{R}^{|E|} \mid M \subseteq E, M \text{ is a matching of } G\}$.

Definition 1.3 (Stable set): A set of vertices such that no two vertices in the set have an edge connecting them is called a *stable set*.

The incidence vector of a stable set S in G , denoted χ^S is a binary vector of length $|V|$, such that, $\chi^S(v) := 1$ if vertex $v \in S$; $\chi^S(v) := 0$ if $v \notin S$.

Definition 1.4 (Stable set polytope): The *stable set polytope* of G denoted by $STAB(G)$ is defined as: $STAB(G) := \text{conv}\{\chi^S \in \mathbb{R}^{|V|} \mid S \subseteq V, S \text{ is a stable set of } G\}$.

Definition 1.5 (Clique): A *clique* is a set of vertices in a graph, any two of which are adjacent to each other.

Definition 1.6 (Odd hole): A chordless cycle with an odd number of vertices (at least 5) is called an *odd hole*.

Definition 1.7 (Complement of a graph): The *complement of a graph* is a graph with the same vertex set, but the edges are those that are not present in G .

Definition 1.8 (Odd antihole): The complement of an odd hole is called an *odd antihole*.

Definition 1.9 (Graph coloring number): The *coloring number* of a graph is the smallest number of colors needed so that, each vertex can be assigned a color with no adjacent vertices receiving the same color.

Definition 1.10 (Induced subgraph): In a graph $G = (V, E)$, the subgraph induced by a set of vertices V' is the graph that consists of V' and those edges both whose ends are in V' .

Definition 1.11 (Perfect graph): A graph $G = (V, E)$ is said to be *perfect* if the coloring number equals the size of the largest clique for every induced subgraph of G .

Definition 1.12 (Intersection graph): The *intersection graph corresponding to a family of sets* is a graph with one vertex for every set, such that two vertices are connected if the corresponding sets intersect.

Definition 1.13 (Interval graph): The intersection graph of a set of intervals on the real line is called an *interval graph*.

Definition 1.14 ((p, q)-colorable graph): A (p, q) -*colorable graph* is a graph whose vertex set can be partitioned into at most p cliques and q stable sets.

Theorem 8 (Strong perfect graph theorem [20]): A graph is perfect if and only if it contains no odd hole and no odd antihole.

Definition 1.15 (Hypergraph): A generalization of a graph in which edges may contain any number of vertices (not just 2) is called a *hypergraph*.

Definition 1.16 (Hyperedge): An edge of a hypergraph is called a *hyperedge*. It can be any arbitrary subset of vertices.

Definition 1.17 (Fractional Weighted Coloring Problem): The *fractional weighted coloring problem* is stated as follows: Given a graph G and a weight $w_v \in \mathbb{R}^+$ for each vertex, minimize $\sum_{i=1}^k \lambda_i$ ($\lambda_i \in \mathbb{R}^+, \forall i$) such that there exist stable sets $\{S_j\}$ of G with $\sum_{i=1}^k \lambda_i \chi^{S_i} = w$, where w is the given weight vector, and χ^S denotes the incidence vector of the stable set S .

Proofs:

Proof: [Proof of Theorem 4] Consider a traffic requirement T consisting of a set of flows and a rate vector. The aim is to decide whether this traffic requirement is within the rate region, with the constraint that fanout-splitting is not allowed.

Construct a new traffic requirement T' from the given one, as follows. For every multicasting input, add a new output and introduce a new unicast flow from the input to the new output. This flow is assigned a rate such that the clique inequality corresponding to the flows at the input becomes an equality. In other words, the net inflow of the multicasting input adds up to exactly 1.

Even if fanout-splitting is allowed for T' , the fact that the net inflow at the input is 1 implies that the fanout must never be split during the schedule, because otherwise, not all flows can be served fully. Any schedule for T' can be restricted to

the flows in T , thus giving a schedule that serves T without splitting the fanout.

This argument shows that T' is achievable with fanout-splitting only if T is achievable without fanout-splitting. This proves the reduction. ■

Proof: [Proof of Lemma 3]

Claim 1: Any point in the convex hull is achievable.

Proof: Let \mathbf{x} be any point in the convex hull. Let J be the number of conflict-free subsets of the multicast flows. Then $\mathbf{x} = \sum_{i=1}^J \phi_i \mathbf{x}_i$, where $\mathbf{x}_i \in R_{A_i}$ and $0 \leq \phi_i \leq 1$; $\sum_i \phi_i = 1$. For every i , since $\mathbf{x}_i \in R_{A_i}$, there is an offline schedule $(\mathbf{a}^{(i)}, \mathbf{S}^{(i)})$ that achieves \mathbf{x}_i , where $\mathbf{S}^{(i)}$ is the sequence of switch states and $a_j^{(i)}$ is the fraction of time, for which the switch should be in state $\mathbf{s}_j^{(i)}$. (Note: Here, the switch state is represented in terms of the incidence vector of the stable set corresponding to the configuration.) Hence,

$$\mathbf{x}_i \leq \sum_j a_j^{(i)} \mathbf{s}_j^{(i)} \quad (9)$$

Consider the schedule: $([\phi_1 \mathbf{a}^{(1)}; \phi_2 \mathbf{a}^{(2)}; \dots; \phi_J \mathbf{a}^{(J)}], [\mathbf{S}^{(1)}; \mathbf{S}^{(2)}; \dots; \mathbf{S}^{(J)}])$, a time-shared strategy among the individual schedules. This schedule achieves \mathbf{x} because $\mathbf{x} = \sum_i \phi_i \mathbf{x}_i \leq \sum_i \sum_j \phi_i a_j^{(i)} \mathbf{s}_j^{(i)}$, using inequality (9).

Claim 2: Any achievable point is in the convex hull.

Proof: Let \mathbf{x} be any achievable point. Then there is a schedule (\mathbf{a}, \mathbf{S}) such that

$$\mathbf{x} \leq \sum_j a_j \mathbf{s}_j \quad (10)$$

The idea here is to group the states in the schedule in terms of which multicasts are being served in the states. Let A_i be the i^{th} conflict-free subset of multicasts. Let $\mathbf{S}^{(i)}$ be a subsequence of \mathbf{S} consisting of states in which the multicasts $\{M_j | j \in A_i\}$ are all being served simultaneously. Let

$$\phi_i = \sum_{\{j | \mathbf{s}_j \in \mathbf{S}^{(i)}\}} a_j, \quad \mathbf{x}_i = \frac{1}{\phi_i} \sum_{\{j | \mathbf{s}_j \in \mathbf{S}^{(i)}\}} a_j \mathbf{s}_j$$

This means \mathbf{x}_i is a convex combination of states with the multicasts in A_i always connected. In other words, this means $\mathbf{x}_i \in R_{A_i}$. Also, substituting in inequality (10), $\mathbf{x} \leq \sum_i \phi_i \mathbf{x}_i$. This shows that \mathbf{x} is in the convex hull of the R_{A_i} 's. ■

Proof: [Proof of Lemma 4] The condition that all multicasts should be simultaneously served throughout the schedule implies that, if any two multicasts overlap, the rate region will be empty, since overlapping multicasts cannot be simultaneously served without fanout-splitting. Also, if the multicasts do not overlap, they get a rate of unity.

If a unicast shares an input or output with any multicast, it cannot be served, since the multicast is always on. The other unicasts do not interact with the multicasts in any way, and can therefore be viewed as a separate problem by themselves. The rate region for the unicasts is then, the matching polytope of the bipartite graph corresponding to unicasts that do not conflict with any multicast. This is given by the two sets of inequalities in the statement of this lemma (see [20]). ■

REFERENCES

- [1] J. Li, "Scheduling algorithms for high speed switches," Ph.D. Dissertation, New Jersey Institute of Technology, May 2001 (available online: NJIT's electronic Theses & Dissertations).
- [2] G. Birkhoff, "Tres observaciones sobre el algebra lineal," *Univ. Nac. Tucuman Rev. Ser. A*, vol. 5, pp. 147–151, 1946.
- [3] J. von Neumann, "A certain zero-sum two-person game equivalent to the optimal assignment problem," *Contrib. to the Theory of Games*, vol. 2, pp. 5–12, Princeton University Press, Princeton, New Jersey, 1953.
- [4] N. McKeown, "The i-SLIP scheduling algorithm for input-queued switches," *IEEE/ACM Trans. Networking*, vol. 7, no. 2, Apr. 1999.
- [5] N. McKeown, "A fast switched backplane for a gigabit switched router," *Business Commun. Review*, vol. 27, no. 12, Dec. 1997.
- [6] N. McKeown, V. Anantharam, and J. Walrand, "Achieving 100% throughput in an input-queued switch," in *Proc. IEEE INFOCOM 1996*, pp. 296–302.
- [7] K. Ross, N. Bambos, K. Kumaran, I. Saniee, and I. Widjaja, "Dynamic scheduling of optical data bursts in time-domain wavelength interleaved networks," in *Proc. IEEE Symposium on High Performance Interconnects*, Aug. 2003.
- [8] K. Ross, N. Bambos, K. Kumaran, I. Saniee, and I. Wadjaja, "Scheduling bursts in time-domain wavelength interleaved networks," *IEEE J. Sel. Areas Commun.*, Optical Communications and Networking Series, Nov. 2003.
- [9] C. S. Chang, W. J. Chen, and H. Y. Huang, "Birkhoff-von Neumann input buffered crossbar switches," in *Proc. IEEE INFOCOM 2000*, pp. 1614–1623, Tel-Aviv, Israel, Mar. 2000.
- [10] C. S. Chang, W. J. Chen, and H. Y. Huang, "On service guarantees for input-buffered crossbar switches: a capacity decomposition approach by Birkhoff and von Neumann," in *Proc. 7th IEEE International Workshop on QoS*, pp. 79–86, London, 1999.
- [11] C. S. Chang, W. J. Chen, and H. Y. Huang, "Birkhoff-von Neumann input buffered crossbar switches for guaranteed-rate services," *IEEE Trans. Commun.*, vol. 49, pp. 1145–1147, July 2001.
- [12] A. K. Parekh and R. G. Gallager, "A generalized processor sharing approach to flow control in integrated service networks: the single node case," *IEEE/ACM Trans. Networking*, vol. 1, no. 3, pp. 344–357, June 1993.
- [13] A. K. Parekh and R. G. Gallager, "A generalized processor sharing approach to flow control in integrated service networks: the multiple node case," *IEEE/ACM Trans. Networking*, vol. 2, no. 2, pp. 137–150, Apr. 1994.
- [14] T. T. Lee, "Nonblocking copy networks for multicast packet switching," *IEEE J. Sel. Areas Commun.*, vol. 6, no. 9, pp. 1455–1467, Dec. 1988.
- [15] B. Prabhakar, N. McKeown, and R. Ahuja, "Multicast scheduling for input queued switches," *IEEE J. Sel. Areas Commun.*, vol. 15, no. 5, pp. 855–866, June 1997.
- [16] M. Andrews, S. Khanna, and K. Kumaran, "Integrated scheduling of unicast and multicast traffic in an input-queued switch," in *Proc. IEEE INFOCOM 1999*, The Conference on Computer Communications, no. 1, pp. 1144–1151, Mar. 1999.
- [17] M. A. Marsan, A. Bianco, P. Giaccone, E. Leonardi, and F. Neri, "Multicast traffic in input-queued switches: optimal scheduling and maximum throughput," *IEEE/ACM Trans. Networking*, vol. 11, no. 3, pp. 465–477, June 2003.
- [18] S. Gupta and A. Aziz, "Multicast scheduling for switches with multiple input-queues," *Hot Interconnects X*, Stanford CA, Aug. 2002.
- [19] C. S. Chang, D. S. Lee, and Y. S. Jou, "Load balanced Birkhoff-von Neumann Switches," in *Proc. IEEE Workshop on High Performance Switching and Routing*, 2001.
- [20] A. Schrijver, *Combinatorial Optimization: Polyhedra and Efficiency*. Springer Verlag, 2003.
- [21] M. Grötschel, L. Lovász, and A. Schrijver, "The ellipsoid method and its consequences in combinatorial optimization," *Combinatorica* 1, pp. 169–197, 1981.
- [22] M. Grötschel, L. Lovász, and A. Schrijver, *Geometric Algorithms and Combinatorial Optimization*, second edition. Springer-Verlag, 1993.
- [23] R. M. Karp, "Reducibility among combinatorial problems," *Complexity of Computer Computations*. New York: Plenum Press, 1972.
- [24] L. G. Khachiyan, "A polynomial algorithm in linear programming," *Soviet Mathematics Doklady*, vol. 20, pp. 191–194, 1979.
- [25] G. Aggarwal, R. Motwani, D. Shah, and A. Zhu, "Switch scheduling via randomized edge coloring," in *Proc. 44th Annual IEEE Symposium on Foundations of Computer Science (FOCS'03)*, pp. 502, 2003.
- [26] H. J. Chao, C. H. Lam, and E. Oki, *Broadband Packet Switching Technologies: A Practical Guide to ATM Switches and IP Routers*. John Wiley & Sons Inc., 2001.

- [27] SIM: a fixed length packet simulator, available at <http://klamath.stanford.edu/tools/SIM/>
- [28] N. McKeown, "Optics inside routers," in *Proc. ECOC 2003*, Rimini, Italy, vol. 2, pp. 168-171, 2003.
- [29] R. Ramaswami and K. N. Sivarajan, *Optical Networks: A Practical Perspective*, 2nd Edition. San Francisco: Morgan Kaufmann Publishers, 2003.
- [30] N. Singhal and B. Mukherjee, "Architectures and algorithms for multicasting in optical WDM mesh networks using opaque and transparent optical cross-connects," in *Proc. IEEE OFC*, Anaheim, CA, Mar. 2001.
- [31] A. Zsigri, A. Guitton, and M. Molnar, "Construction of light-trees for WDM multicasting under splitting capability constraints," in *Proc. 10th IEEE International Conference on Telecommunications*, New York, pp. 171-175, 2003.
- [32] B. Mukherjee, *Optical Communication Networks*. New York: McGraw-Hill, 1997.
- [33] L. H. Sahasrabudhe and B. Mukherjee, "Light-trees: optical multicasting for improved performance in wavelength-routed networks," *IEEE Commun. Mag.*, vol. 37, pp. 67-73, 1999.
- [34] M. Ali, "Optimization of splitting node placement in wavelength-routed optical networks," *IEEE J. Sel. Areas Commun.*, issue on network support for multicast communications, vol. 20, no. 8, pp. 1571-1579, Oct. 2002.
- [35] C. Caramanis, M. Rosenblum, M. X. Goemans, and V. Tarokh, "Scheduling algorithms for providing flexible, rate-based, quality of service guarantees for packet-switching in Banyan networks," in *Proc. 38th Annual Conference on Information Sciences and Systems*, Princeton, NJ, pp. 160-166, 2004.
- [36] L. Tassiulas and A. Ephremides, "Stability properties of constrained queueing systems and scheduling policies for maximum throughput in multihop radio networks," *IEEE Trans. Automatic Control*, vol. 37, no. 12, pp. 1936-1948, 1992.
- [37] V. E. Alekseev and V. V. Lozin, "Independent sets of maximum weight in (p, q) -colorable graphs," *Discrete Mathematics*, vol. 265, no. 1, pp. 351-356, 2003.



Jay Kumar Sundararajan received the B. Tech. degree in electrical engineering from the Indian Institute of Technology, Madras, India, in 2003, and the S.M. degree in electrical engineering and computer science from the Massachusetts Institute of Technology, Cambridge, USA, in 2005.

He is currently pursuing a Ph.D. degree in electrical engineering and computer science at the Massachusetts Institute of Technology, Cambridge, USA. His research interests include network coding, information theory and network algorithms.



Supratim Deb received the B.E. degree in electrical and electronics engineering from the Birla Institute of Technology and Science, Pilani, India, in 1997, the M.E. degree in telecommunication from the Indian Institute of Science, Bangalore, in 1999, and the Ph.D. degree in electrical engineering from the University of Illinois at Urbana-Champaign, in 2003.

He was a Postdoctoral Research Associate at the Massachusetts Institute of Technology, Cambridge, from August 2003 to January 2005. At present, he is a Member of the Technical Staff at Alcatel-Lucent Bell Laboratories, Bangalore, India. His research interest lies broadly in the area of network algorithms.



Muriel Médard is an Associate Professor in the Electrical Engineering and Computer Science at MIT and the Associate Director of the Laboratory for Information and Decision Systems. She was previously an Assistant Professor in the Electrical and Computer Engineering Department and a member of the Coordinated Science Laboratory at the University of Illinois Urbana-Champaign. From 1995 to 1998, she was a Staff Member at MIT Lincoln Laboratory in the Optical Communications and the Advanced Networking Groups. Professor Médard received B.S. degrees in EECS and in Mathematics in 1989, a B.S. degree in Humanities in 1990, a M.S. degree in EE 1991, and a Sc D. degree in EE in 1995, all from the Massachusetts Institute of Technology (MIT), Cambridge. She serves as an Associate Editor for the Optical Communications and Networking Series of the *IEEE Journal on Selected Areas in Communications*, as an Associate Editor in Communications for the *IEEE Transactions on Information Theory* and as a Guest Editor for the Joint special issue of the *IEEE Transactions on Information Theory* and the *IEEE/ACM Transactions on Networking* on Networking and Information Theory. She has served as a Guest Editor for the *IEEE Journal of Lightwave Technology* and as an Associate Editor for the *OSA Journal of Optical Networking*.

Professor Médard's research interests are in the areas of network coding and reliable communications, particularly for optical and wireless networks. She was awarded the IEEE Leon K. Kirchmayer Prize Paper Award 2002 for her paper, "The Effect Upon Channel Capacity in Wireless Communications of Perfect and Imperfect Knowledge of the Channel," *IEEE Transactions on Information Theory*, Volume 46 Issue 3, May 2000, Pages: 935-946. She was co-awarded the Best Paper Award for G. Weichenberg, V. Chan, M. Médard, "Reliable Architectures for Networks Under Stress", Fourth International Workshop on the Design of Reliable Communication Networks (DRCN 2003), October 2003, Banff, Alberta, Canada. She received a NSF Career Award in 2001 and was co-winner 2004 Harold E. Edgerton Faculty Achievement Award, established in 1982 to honor junior faculty members "for distinction in research, teaching and service to the MIT community." She was named a 2007 Gilbreth Lecturer by the National Academy of Engineering. Professor Médard is also House Master at Next House.