Extending the Birkhoff-von Neumann Switching Strategy to Multicast Switching

by

Jay Kumar Sundararajan

B. Tech., Electrical Engineering (2003)
Indian Institute of Technology, Madras

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Science in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 2005

Author . . . .
Department of Electrical Engineering and Computer Science
January 28, 2005

Certified by . . . .
Muriel Médard
Esther and Harold E. Edgerton Associate Professor
Thesis Supervisor

Certified by . . . .
Supratim Deb
Post Doctoral Associate
Thesis Supervisor

Accepted by . . . .
Arthur C. Smith
Chairman, Department Committee on Graduate Students

Extending the Birkhoff-von Neumann Switching Strategy to Multicast Switching

by

Jay Kumar Sundararajan

Submitted to the Department of Electrical Engineering and Computer Science
on January 28, 2005, in partial fulfillment of the
requirements for the degree of
Master of Science in Electrical Engineering and Computer Science

Abstract

The Birkhoff-von Neumann (BVN) strategy for offline switching does not support multicast, as it considers only permutation-based switch configurations. This thesis extends the BVN strategy to multicast switching. Using a graph theoretic model, we show that the capacity region for a traffic pattern is precisely the stable set polytope of the pattern's "conflict graph", in the no-fanout-splitting case. We construct examples to show that, if dynamic fanout splitting is excluded, there is no clear winner in terms of rate region among various fanout splitting strategies.

The problem of deciding whether a given set of rates is achievable in a multicast switch is also addressed. We show that, in general, the problem is equivalent to the membership problem for the stable set polytope of a graph, and is therefore NP-hard. We also prove that the problem is NP-hard for the case that splitting of the set of destinations, or fanout, is allowed. However, in the no-splitting case, it is polynomial time solvable when the number of multicast flows in the $N \times N$ switch is $O(logN)$. The algorithm naturally leads to a schedule to serve the flows in a stable manner, if the rates are achievable. For an arbitrary number of multicasts, we show that, computing the offline schedule is equivalent to fractional weighted graph coloring which takes polynomial time for perfect graphs. We present several types of traffic patterns whose conflict graphs are perfect.

[18] proposed a simple online algorithm called i-SLIP based on parallel iterative matching, for online unicast scheduling. We propose an online algorithm for multicast, based on i-SLIP and the conflict graph idea, and compare them with ESLIP([19]) and the copy-and-use-i-SLIP strategy, through simulations.

Thesis Supervisor: Muriel Médard
Title: Esther and Harold E. Edgerton Associate Professor

Thesis Supervisor: Supratim Deb
Title: Post Doctoral Associate

Acknowledgments

I thank Prof. Muriel Médard, my research advisor, for all her invaluable support and guidance in my life at MIT. Besides teaching me the right approach to research, she has been a great source of inspiration. I thank her for her patience and care during the course of this work.

I thank Dr. Supratim Deb, my co-supervisor, whose assistance has been vital in the completion of this work. My association with him has definitely helped me learn several skills and new approaches in tackling a research problem.

I thank my friends Vinod Vaikuntanathan and Raghavendran Sivaraman from MIT, who helped me greatly through useful discussions related to this work.

I thank my mother Mrs. Vasantha Sundararajan for her constant love, support, guidance and encouragement, which have been vital in my entire life.

My thanks also go to all my friends, relatives and well-wishers I have not mentioned here, who have guided and helped me on various occasions. I thank the universe for bringing such wonderful people into my life.

This work was supported by the following grants:

- NSF Award number ANI - 0335217 - All Optical Networks
- NSF Subcontract to University of Illinois Sub Award Number 2003-07082 - ITR/SY: High Speed Wavelength Agile Optical Network
- NSF Award number CCR - 0093349 - Career

The financial assistance is greatly appreciated.

Contents

List of Figures

Chapter 1

Introduction

The problem of switch scheduling is well studied in the literature. It is well known that output queued (OQ) switches can achieve 100 % throughput and can also provide quality of service (QoS) guarantees. However, OQ switches operate at N times the line rate (where N is the switch size). Therefore, because of the high speedup and memory bandwidth requirements, this approach does not scale with the size of the switch. An input queued (IQ) switch architecture, with virtual output queues to avoid head of line blocking, is a scalable option, since this operates at the line rate. McKeown et al. [3] 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, though the delay performance is worse. A detailed survey of the various extensions and simplifications of the MWM algorithm, as well as other strategies, is presented in [11]. One major drawback of MWM-based approaches is that they provably do not provide cell delay guarantees. There are several schemes that address this issue. One such scheme is the Birkhoff-von Neumann switch proposed by Chang et al. [5].

The Birkhoff-von Neumann (BVN) switch provides 100 % throughput for all non-uniform traffic, and also gives deterministic cell delay guarantees for certain types of traffic. The BVN approach is based on a theorem by Birkhoff [1] and von Neumann [2], which says that, any doubly stochastic matrix[1] can be expressed as a convex combination of per-

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

mutation matrices. Thus, any achievable rate matrix representing required rates for every input-output pair, can be first converted to a doubly stochastic matrix, and then decomposed into permutation matrices. A permutation matrix naturally corresponds to a switch connection state. Thus, a convex combination of permutation matrices leads to 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.

A natural extension of the unicast scheduling problem is the multicast case, where connections are no longer constrained to be point-to-point, but may have multiple destinations. Initial work on the problem of scheduling multicast in an IQ switch was based on the copy strategy – copy networks were designed as a separate stage before the switching fabric itself [4], in order to replicate multicast cells at the inputs and treat them as unicast cells. However, this has some disadvantages. Since each copy is equivalent to a new cell, the bandwidth available to other traffic in the switch is reduced. Besides, a memory speedup is needed, because, in the time it takes for a single packet to arrive, the memory has to store multiple copies of the packet.

This leads to the realization that, *intrinsic multicast capability*[2] in the switching fabric may help to reduce the cost of multicast traffic management. Prabhakar et al. [17] consider an $M \times N$ input-queued switch with intrinsic multicast capabilities. Only one queue is maintained per input and only multicast flows are assumed to be present. The work proposes a scheduling algorithm which allows fanout-splitting and tries to concentrate the residue on as few inputs as possible. An important idea proposed in [8] and [19] is 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. The paper [8] also proves that the problem of finding the optimal multicast switching schedule is NP-hard for several variants of the problem. This paper does not assume knowledge of the rates of multicast flows. Marsan et al. [20] proved that, unlike in the unicast case, the rate region of IQ switches is strictly smaller than the rate region of OQ switches even when intrinsic multicast is allowed. This means, all admissible traffic patterns may not be sustainable in the case of multicast, in an IQ switch. This paper also gave the optimal online multicast

[2]The ability to simultaneously transfer a cell to multiple outputs using simultaneous switching paths

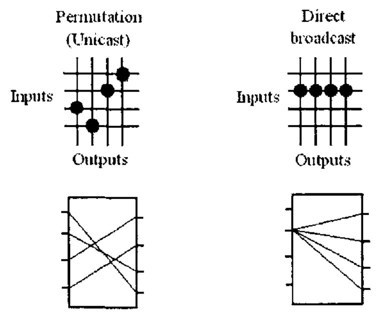


Figure 1-1: Switch connection states : Permutation and Direct Multicast

scheduling algorithm, and showed that the achievable rate region is precisely the convex hull of departure vectors. However, they did not give an explicit characterization of the rate region.

All the earlier work focuses on the case when fanout-splitting is allowed, and the manner in which the fanout is split can be changed dynamically. This approach requires an exponential number of VOQs at each input. The other option is the no-splitting approach, in which the fanout is never split. The number of VOQs at each input is not exponential in the switch size in this case. In this thesis, we address the question of how to compute an offline schedule given the rates, in a manner akin to BVN. We focus on the no-splitting approach, and our results can be extended to the static splitting case (see Section 2.1). [18] proposed a simple online algorithm called i-SLIP based on parallel iterative matching, for online unicast scheduling. We also propose a heuristic online algorithm for multicast, based on the i-SLIP algorithm.

If we want to be able to provide rate and cell delay guarantees for multicast, we must adopt an approach like the BVN switch. Several issues arise if we attempt to extend the BVN approach to multicast. The first, most immediate one, is that the BVN strategy considers only permutation-based states of the switch and disallows direct multicast states. Fig. 1-1 shows a crossbar switch that supports intrinsic multicast and can be used to multicast a cell directly.

The second issue is the question of fanout-splitting. Fanout refers to the set of destinations of a multicast flow. As mentioned earlier, one way to handle multicasts is to replicate the cell as many times as the number of destinations and then treat the copies as unicast cells. The other extreme is direct multicast with no fanout-splitting. Here, the multicast cell is transferred to all its outputs in one time slot through simultaneous switching paths. In between these two extremes, there is a whole range of intermediate options, where the fanout is split partially. Multiple copies of the multicast cell are generated and in any one slot, one copy is transferred to a subset of the fanout which has not already received the cell. These options are described in detail and compared in the next chapter.

1.1 Goals

This thesis explores the problem of extending the Birkhoff-von Neumann switching strategy to the general scenario where both unicast and multicast flows need to be served in the same switch. The main goals of this work are to provide answers and insights to the following questions:

1. How can one extend the Birkhoff-von Neumann approach to multicast case when dynamic fanout splitting is not allowed? In other words, what is the algorithm to decompose the rate requirements for multicast flows, into a sequence of switch configurations which would specify an offline schedule?

2. Given a rate requirement for various flows – unicast and multicast, how can one decide whether it is within the rate region for each of the above strategies? Is it possible to find an explicit characterization of the rate region in the various cases? Note that, in the unicast case, the admissibility conditions (i.e. no input and no output is overbooked) completely characterize the rate region.

3. How does fanout-splitting affect the rate region? Clearly, dynamic splitting subsumes other cases, since it is the least constrained. But, suppose dynamic splitting is not allowed, then is it better to copy completely, or to go for the no-splitting option, or something in between?

4. How to 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 or when issues of speed preclude even moderate polynomial time algorithms, such as bipartite matching or related algorithms.

1.2 Implications for Optical Switching

1.2.1 Optics inside Routers

High speed routers today 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 [25]. There are two options here. One is the all-optical switch, with fiber delay-line buffers used to buffer packets. The paper by McKeown [25] suggests that, for high-speed Gb/s routers, the all-optical approach is 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 are quite complicated and are not scalable. 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. Another advantage of optical switching fabric is the intrinsic multicast capability. It is possible to send a beam of light to multiple outputs simultaneously in an optical switching fabric, using power splitters. The optimal solution will be to exploit the strength of the two technologies. This means, do the buffering in the electronic domain, and use an optical switching fabric.

An important issue with optical switching fabrics is that they can be slow to reconfigure. For instance, MEMS switches take about 10ms to reconfigure. Therefore, an offline scheduling algorithm akin to the Birkhoff-von Neumann algorithm, which we study in this work, is particularly useful for switches with optical fabrics. If the sequence of configu-

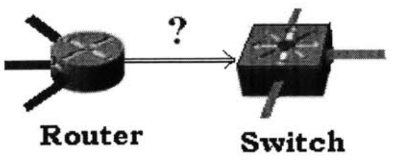


Figure 1-2: How should a router exploit the optical switching fabric?

rations and associated fractions of time are known beforehand, then we can minimize the number of reconfigurations by scheduling the states in contiguous blocks of time.

1.2.2 Multicast Switching in Optical Networks

In order to implement multicast in an optical network, the network layer multicast protocol running in the router has to create the multicast tree. To realize this physically, the router has one of two options, as explained in [26] and [27] – 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 itself for multicasting using all optical cross-connects. Note that this requires power splitters along with amplifiers to compensate for the resulting losses. The other approach, called the *opaque* approach, is to convert the light to the electronic domain, switch using crossbar switches and finally revert to the optical domain.

The second option is to use the network or IP layer multicasting with physical layer unicasting, which avoids the use of O/E/O converters and the splitting ability of switches. Here the router replicates the packets and unicasts different copies to the various destinations.

At first, it may seem that it is always better to multicast directly, rather than make multiple copies and then unicast them. However, in this work, we show that, if dynamic fanout 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.

1.3 Main Contributions and Thesis Outline

The main contributions of this work are described in this section, along with the overall outline of the thesis.

This thesis is a step towards understanding the rate region for multicast flows in a switch. It is known that the maximum normalized throughput for IQ switches under uniform multicast traffic is always less than one, and that it depends on the multicast traffic distribution [20]. The rate region also depends on the exact traffic pattern that is being served. We develop a graph theoretic model that helps address the problem of characterizing the rate region for a given traffic pattern that includes multicast and unicast flows.

- Chapter 2 covers some relevant background and formally states the problem addressed in this thesis. It contains a brief summary of the motivation and details of the Birkhoff-von Neumann switch. It also describes several strategies for serving multicast traffic in a switch and compares them in various respects. We have shown through an example that there is no clear winner between copying and static splitting/no-splitting strategies, in terms of the rate region.

  We introduce a graph theoretic framework that helps us to study the achievable rate region in a switch with multicast flows. Using this model, we show that the rate region of the no-splitting case is the stable set polytope of a "conflict graph", and that, in general, there is no explicit polynomial sized characterization for this region. This provides an entirely new way of approaching the problem, and helps us use many results from graph theory.

- In Chapter 3 we show that, deciding whether a given set of rates is achievable is NP-hard in general. However, for the no-splitting case with a moderate number of multicast flows, we provide a polynomial time algorithm that decides whether a given rate requirement vector is achievable or not. More precisely, the algorithm is polynomial in the size of the switch (N) when the number of multicast flows is $O(logN)$ or smaller. Our algorithm naturally leads to an offline schedule.

- In addition, we show that, the decomposition of the rate requirements into switch connection states (as in BVN) is equivalent to the *fractional weighted coloring problem*. This leads to an offline scheduling algorithm for the no-splitting case, which is polynomial time for the case of perfect conflict graphs with a polynomial number of multicast flows. We also explicitly characterize the rate region for the special case when the conflict graph is *perfect*. We give various examples when such a scenario may arise.

- In Chapter 4 we study the performance of two online heuristic algorithms for multicast using the ideas of the unicast algorithm *i*-SLIP [18]. One is inspired by the conflict graph formulation and is applicable to the no-splitting case. The other one is a modification of ESLIP [19] for the case when there is more than one virtual output queue for multicast at each input, and can be used when dynamic fanout-splitting is allowed. We compare these algorithms in terms of delay and throughput simulations.
- Finally in Chapter 5, we present the conclusions and inferences of our work. We also discuss possible lines of future work related to this problem.

Chapter 2

Background and Problem Formulation

2.1 Strategies for Multicast

In this work, the term multicast flow refers to a set of packets that originate at a given input and are destined to reach a given set of outputs in a switch, called the *fanout set*. In other words, an input and a subset of the outputs uniquely identify a multicast flow. Head-of-line (HOL) blocking can be prevented if and only if each input maintains a separate VOQ for each multicast flow originating at that input.

There are many strategies to support multicast traffic in an input-queued (IQ) switch. It is assumed that each packet is broken into fixed-size cells on the input side, which are reassembled on the output side. Each multicast cell can be handled in any of the following ways. Note that different queue policies may be appropriate for the different cases.

1. **Copying:** 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 corresponding to each of the outputs in its fanout. This is equivalent to viewing the multicast as a collection of unicast flows. It requires only N VOQs per input in an $N \times N$ switch and is easily implemented in switches which use *internal copy networks* or *recirculating lines*. Copy networks are designed as a separate stage before the switching fabric [4]. However, this has some disadvantages. As each copy is equivalent to a new cell, the bandwidth available to other traffic in the switch comes down.

2. **No-splitting:** This is the other extreme, where a multicast cell is sent to all outputs in its fanout in a single slot. Each input needs to maintain a separate VOQ for every multicast flow. Moreover, the switch should have intrinsic multicast capability. Fig. 1-1 shows a crossbar fabric architecture that supports intrinsic multicast.

3. **Fanout-splitting:** There is an entire spectrum of strategies which generalize the above extremes by partially splitting the fanout of the multicast flows. 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. Again, the switch needs to have intrinsic multicast capability. This strategy can be implemented in two ways:

   - Static splitting: Here, the multicast traffic pattern is assumed to be known before hand. The manner in which each multicast flow's fanout will be split is decided offline, and is kept fixed. All cells of a flow are split in the same way. This is like replacing the original multicast with 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. Note that copying and no-splitting are special cases of static splitting – in both cases, the rule for splitting is decided beforehand. Note that it is possible to view static splitting as an instance of no-splitting, with the set of flows chosen as the split flows. So, all results derived for no-splitting strategy hold for the static splitting case also.

   - Dynamic splitting: When a multicast cell arrives at an input, it is transferred to some subset of its fanout. If has not reached its complete fanout, then it is re-enqued into the VOQ corresponding to the remaining part of the fanout. For this strategy, each input needs to maintain a separate VOQ for every possible subset of the fanout. In this approach, the way in which a multicast flow's fanout is split can vary from cell to cell.

2.1.1 Implementation Issues

Dynamic splitting represents the least constrained approach, since different cells of the same flow can be split differently. It is known that, in the online case, dynamic fanout-splitting gives better throughput than no-splitting [19]. However, this benefit comes at the cost of maintaining an exponential number of VOQs at each input, even if the traffic pattern is known beforehand. This is because, the way flows are split is not known beforehand. Thus, after a particular slot, the part of the fanout that remains to be served could be any arbitrary subset of the original fanout. Therefore, each input must maintain a separate VOQ for every possible subset of the fanout, thereby resulting in an exponential number of queues. As compared to this, *the static splitting approach requires a much smaller number of queues.* Since the split of the fanout is known beforehand, it is enough if the input maintains a single queue for every split flow.

It is possible that different inputs may follow different strategies from the above list, based on the knowledge of the traffic rates coming into them. Here is an example. Consider a case when there is only one multicast flow at each input besides unicast flows. Each input is allowed to choose independently either the copy strategy or the no-splitting strategy for all flows, in order to keep things simple and localized. Copying increases the unicast load at an input. So one distributed local heuristic is that those inputs which already have a high unicast load, may decide beforehand not to use the copy strategy.

An Example : The 2×2 Case

While it might, at first, appear that copying cannot outperform direct multicast (static or no splitting), we shall see that neither technique dominates the other. For a 2×2 switch, the rate region is shown in Fig. 2-1 in terms of achievable multicast rates, for a fixed value of the unicast rates. It brings to light an interesting phenomenon. Consider the case when dynamic splitting is not allowed. If the unicast demands are not balanced at the two inputs, then we could do better by copying at the input which has less unicast traffic. This is surprising at first; copying packets at the input is usually considered wasteful, since it uses up more time slots than if we directly multicast them. For example, we see that, if we copy

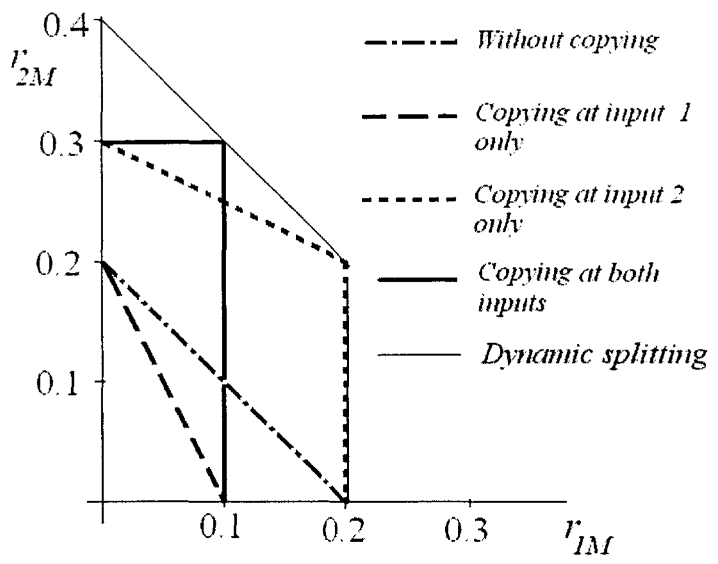


Figure 2-1: The rate region of the multicast flows in a 2×2 switch, with and without copying, with unicast rates fixed as: $r_{11} = r_{12} = 0.4$, $r_{21} = r_{22} = 0.2$.

only at input 2 and do a direct multicast at input 1 (the busier input), then we get a rate region that covers certain points the direct multicast strategy cannot achieve. This means that the decision of whether to split the fanout will have to be made for each flow based on the rates. For points that are within the rate region of more than one copy strategy, the choice of the strategy determines the delay performance. In particular, that strategy in which the required rates are closer to the edge of the capacity region will, in general, lead to longer delay.

The above strategies can, in general, be implemented *online* or *offline*. (The only exception is the static splitting case, which can be done only offline.) In this work, the terms online and offline are used in the following sense. Online implementations look at the current state of the queues in each slot, and decide the switch configuration for that slot. Offline implementations assume knowledge of the average traffic rates for each flow, and generate a decomposition of the rate matrix into a sequence of switch states, which gives an offline schedule. In other words, the switching schedule computation is done offline using knowledge of only the traffic rates. The Birkhoff-von Neumann switch achieves this for the unicast case ([5],[6],[7]).

2.2 Birkhoff-von Neumann Switches

The Birkhoff-von Neumann (BVN) switching algorithm [5] is a switching algorithm that provides rate guarantees as well as cell delay guarantees for input-queued crossbar switches, while achieving 100% throughput. The gist of the BVN approach is to decompose a set of real demands into a convex combination or permutation matrices, which naturally correspond to switch configurations. The approach is briefly described here.

Consider an $N \times N$ input-queued crossbar switch. Let $R = (r_{ij})$ be the required rate matrix, such that, r_{ij} represents the rate requirement from input i to output j. The earlier work on Birkhoff-von Neumann switches [5], [6], [7] has shown that, if:

$$\sum_{i=1}^{N} r_{ij} \leq 1, \quad \forall j = 1, \dots, N \tag{2.1}$$

$$\sum_{j=1}^{N} r_{ij} \leq 1, \quad \forall i = 1, \dots, N \tag{2.2}$$

(i.e. the rate matrix is doubly sub-stochastic), then, there exist positive numbers ϕ_k and permutation matrices P_k ($k = 1, \dots, K$, 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 performs switch scheduling by setting the crossbar connections as specified by P_k for a fraction of time equal to ϕ_k. This is done for all k from 1 to K where K is the number of such permutation matrices.

To prove that the rate guarantees are indeed met, one may use the BVN theorem [1], [2], which states that any $N \times N$ doubly stochastic matrix (one whose row and column sums are all 1) can be expressed as a convex combination of no more than $(N^2 - 2N + 2)$ permutation matrices. The first step is to replace the possibly sub-stochastic rate matrix by a doubly stochastic matrix which majorizes the original rate matrix. This step is quite simple and the algorithm is given in [5]. An alternate, less complex, algorithm is also given in [11], called the *weighted rate filling algorithm.*

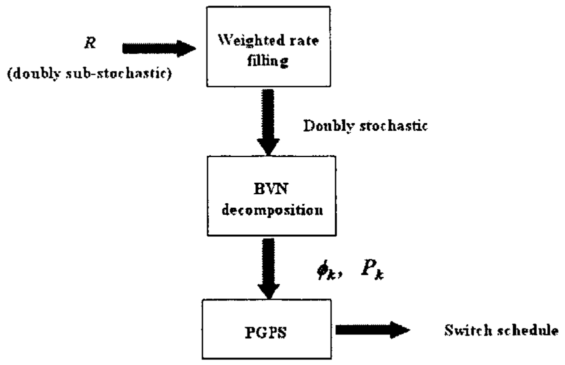


Figure 2-2: The BVN switching algorithm

Once the doubly stochastic matrix has been obtained, it 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 identical to the problem of finding perfect matchings in a bipartite graph. The matching problem and its associated geometric interpretation will be discussed in Section 2.5. Note that, the matching problem is solved offline and has a complexity of $O(N^{2.5})$, where N is the size of the switch.

Having obtained the decomposition into permutation matrices offline, the scheduling itself may be done online, according for instance to an algorithm [5] based on the Packetized Generalized Processor Sharing [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 the BVN is to randomly pick 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. The sequence of steps in a BVN switch is indicated in Fig. 2-2

Using the BVN algorithm as a starting point for resource allocation, other algorithms have been developed to take into account the current occupancy of the queues also, in various ways. This problem is addressed in [9], where the enhanced BVN switch is proposed.

The BVN approach discussed above is only for unicast flows as it considers only permutation based switching states. An extension of the BVN approach is given in [10] where a two-stage architecture is proposed with a load balancer preceding the actual scheduler.

This switch achieves 100 % throughput for unicast and multicast with fanout-splitting, under certain assumptions on the input traffic.

The remaining part of this chapter describes the switch model used and gives a formal statement of the problem addressed in this work. A graph-theoretic framework is developed in terms of "conflict graphs". This framework leads to a better understanding of the underlying issues, and also allows us to use results from graph theory.

2.3 Switch Model

We consider an input-queued switch with a non-blocking switch fabric such as the crossbar fabric. As mentioned in an earlier section, in order to prevent head-of-line blocking, each input maintains a separate queue for every flow it handles. There is no restriction on the queue lengths.

The switching fabric introduces the following constraints in the problem. An input can transmit only one cell in a single time slot. The cell may reach any number of outputs at the same time, i.e. intrinsic multicast is allowed. An output may receive only one cell in a time slot from any one input only. In short, two cells that are simultaneously being served in a time slot cannot clash at the input or at any output. We consider switches with no speedup. This means, there can be at most one arrival to any input in a single time slot, and in this time, at most one cell can be removed from any input, and at most one cell can be delivered to any output.

An important assumption, which makes this work different from earlier work, is that the scheduler knows beforehand the multicast traffic pattern as well as the average rates of the flows in the pattern. The same assumption is made in the work on the Birkhoff-von Neumann switch, such as [5], for unicast.

We now briefly discuss the i-SLIP algorithm proposed in [18].

2.3.1 The i-SLIP algorithm

The i-SLIP algorithm, proposed in [18], is a simple and easily implementable algorithm for online unicast scheduling. Its worst case complexity is linear in the number of inputs/outputs in the switch, though it usually runs in logarithmic time. i-SLIP is an iterative algorithm. Each iteration involves three stages — request, grant and accept. In the request step, each input sends a request to the outputs for which it has cells. Each output maintains a pointer which takes on values of the input number, in a round-robin manner. In the grant step, each output selects the request that appears next in a round-robin schedule, starting from the current position of the pointer. The input also maintains a pointer which takes on values of the output number, in a round-robin manner. In the accept step, each input selects one of the grants it has received — the one which appears next in a round-robin schedule starting from the present position of that input's pointer. Both the input and output side pointers are updated only if the input accepts a grant in the first iteration.

2.4 Problem Statement

The goals of our work were discussed in Section 1.1 in an informal way. In this section, we restate the problems in a more formal way. We focus on the no-splitting case first. The term *flow* refers to a set of packets (cells) which originate at a certain input and have to reach the same set of outputs. In other words, a flow is characterized by its input and set of outputs it is destined for. Let $F = (f_1, f_2, \ldots)$ be the set of flows required to be supported by the switch. Let the corresponding rates be $r_1, r_2, \ldots$. These rates are assumed to be normalized with respect to the incoming (or outgoing) line rate. The flows could be unicast or multicast. A set of flows that can be served simultaneously in the switch (i.e. without conflicts in the inputs or the outputs) defines a valid *switch connection state.*

The problems we address in this work are formally stated below:

1. The first problem is to find a schedule that serves these flows in a stable manner using a no fanout-splitting strategy, without allowing the input queues to grow unboundedly. More formally, we need to develop an algorithm to find a sequence of positive fractions ϕ_k's and switch connection states S_k's such that if the switch is allowed to remain in state S_k for a fraction of time ϕ_k, then every flow's rate requirement is

satisfied, i.e.:

$$r_i \le \sum_{\{k: f_i \in S_k\}} \phi_k \quad \forall i \tag{2.3}$$

2. A natural sequel is the problem of explicitly characterizing the rate region. The aim is to find a set of necessary and sufficient conditions on the rates $(r_1, r_2, \ldots)$, so that such that a sequence (S_k, ϕ_k) exists. Suppose an explicit characterization is not possible, then a related question is to decide whether a given set of rates $(r_1, r_2, \ldots)$ is achievable, i.e. whether a schedule exists for this set of rates.

3. If dynamic splitting is excluded because of implementation issues, then how do the remaining strategies, namely, no-splitting, static splitting and complete fanout splitting, compare in terms of achievable rate region. Is there a single winner?

4. We also address the case when the traffic pattern and the rates are not known beforehand. This is basically the online setting. The problem is to extend the unicast heuristic online algorithms such as i-SLIP to handle multicast flows. This problem has been looked at before and a few solutions, such as ESLIP, have been proposed in the literature.

2.5 The Unicast Rate Region

In the case of unicast traffic, the capacity region is precisely the perfect matching polytope of a complete bipartite graph, because any point inside the polytope can be described as a convex combination of matchings, which immediately gives a schedule. The matching polytope of the complete bipartite graph has been characterized completely [14]. It turns out to be precisely the BVN rate region described earlier in Eqn. (2.1) and (2.2).

Let x_e denote the weight of edge e. Then, the perfect matching polytope of a bipartite graph $G = (V_1, V_2, E)$ is given by:

$$M(G) = \{\chi \in \mathbb{R}^{|E|} | x_e \ge 0 \ \forall e \in E, \sum_{e \in \delta(v)} x_e = 1 \ \forall v \in V_1 \cup V_2\} \tag{2.4}$$

where $\delta(v)$ is the set of edges incident to vertex v.

This set of constraints is the same as specifying that the rate matrix should be doubly stochastic. The Birkhoff-von Neumann decomposition into permutation matrices is thus simply the matching decomposition of the bipartite graph.

If we try to extend this model to the multicast case, it becomes cumbersome, because the new 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 in the next section.

2.6 The Multicast Rate Region

2.6.1 Conflict Graph Formulation

Definition 2.6.1 (Conflict graph) *The conflict graph for a given traffic pattern is defined thus:*

Define graph $G = (V, E)$ *:*

V *= set of all flows to be served*

$E = \{(v_i, v_j) | \textit{flows } i, j \textit{ cannot coexist in a valid switch configuration}\}$.

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. For example, two flows from different inputs to the same output cannot co-exist and so their vertices would be connected. This kind of a graph brings out the connection constraints in the switch described in Section 2.3. Fig. 2-3 shows an example of a traffic pattern and the corresponding conflict graph.

Now, a valid switch configuration consists of a set of flows that can co-exist. In the conflict graph, such flows corresponds 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*[1] of the conflict graph of the flows. This arises from the fact that there is a one-to-one correspondence between a sequence of valid switch

[1]The convex hull of incidence vectors of stable sets of a graph is called the *stable set polytope*

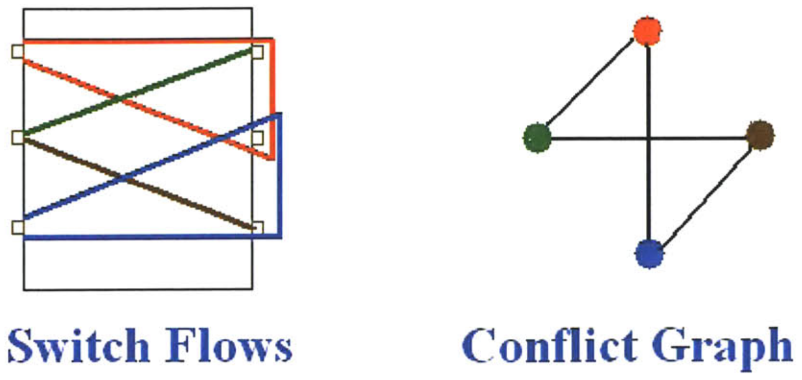


Figure 2-3: An example of a traffic pattern and its conflict graph

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.

The discussion above immediately leads to the following theorem:

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

Note that 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 [14]). So one would expect that there are polynomial time algorithms to decide achievability and even find a schedule, in the case of unicast. This is indeed true, and the BVN switch is in fact based on this idea.

For a perfect graph, the stable set polytope is completely characterized by the clique inequalities. (For the definition of *clique*, see Appendix A). The cliques in the conflict graph of a unicast traffic pattern correspond to of a set of flows that begin at the same input or flows that terminate at the same output. The condition that the rate matrix should be doubly sub-stochastic thus gives the same rate region as the stable set polytope of the conflict graph.

Appendix B discusses results on the stable set polytope of special classes of graphs. However, for a general graph, an explicit characterization of the polytope is not known so far. In view of this fact, we address the question of deciding the achievability of a given set of rates, in the next chapter.

Chapter 3

Complexity, Achievability and Offline Algorithms

In Chapter 2, a graph theoretic framework was developed to study the multicast scheduling problem. The rate region was shown to be the stable set polytope of the conflict graph for a given traffic pattern. In this chapter, we investigate the problem of deciding whether a given traffic pattern is achievable or not. Given a set of unicast and multicast flows and their average arrival rates, is it possible to say whether this traffic pattern is sustainable, i.e. whether this rate point is within the achievable rate region? Note that an explicit characterization for the stable set polytope of a general graph is not known.

We show that the problem of deciding achievability is NP-hard in general, whether fanout-splitting is allowed or not. In the no-splitting case, the problem is shown to be fixed-parameter tractable in the number of multicast flows. This means that, if the number of multicast flows is of the order of the logarithm of the number of input/output ports, then there is a polynomial time algorithm to decide achievability. The algorithm we propose naturally yields a schedule if the rate point is within the achievable region. These results readily generalize to the static splitting case, for a given splitting policy.

In this chapter we also address the question of finding the scheduling algorithm given a traffic pattern and a set of rates for the flows. This problem is shown to be a fractional weighted graph coloring problem. If the conflict graph is a perfect graph, this problem can be solved in polynomial time. Besides, for a perfect graph, an explicit characterization of the stable set polytope is available in the literature. We apply these results to derive the rate region of several special cases of traffic patterns, where the conflict graph is perfect.

3.1 Deciding Achievability is NP-hard in general

Lemma 1 *Given a general graph G, there exists a multicast traffic pattern in a $|V(G)| \times |V(G)|$ switch, with G as the conflict graph, corresponding to the no-splitting case.*

Proof: For any given graph G, there exists a family of sets such that G is the intersection graph of the family. Compute this family of sets, say $\{S_1, S_2, \ldots, S_{|V(G)|}\}$. Now, create a traffic pattern in a $|V(G)| \times |V(G)|$ switch where, 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 clashes at the inputs, and since G is the intersection graph of the fanout sets. $\square$

Theorem 2 *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, we can come up with 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.

One way to solve the maximum weighted stable set problem is to maximize a linear function over the stable set polytope, the convex hull of the incidence vectors of all stable sets of the graph. 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 a way to answer the membership problem, i.e., if there is an oracle that decides whether a given point is within a convex polytope or not, then there is an efficient way to optimize over that polytope. This fact is based on the result given in [15] that the weak membership problem is at least as hard as the weak optimization problem. The strong membership problem is at least as hard as the weak membership problem for the stable set polytope (see [15]) for further details).

To summarize, the achievability question is essentially the strong membership problem for the stable set polytope. Optimization over the stable set polytope can be reduced to this strong membership problem. 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 [22]. This proves the theorem. □

3.2 Fanout-Splitting Case is as hard as No-Splitting

This section proves that, if an efficient algorithm exists to decide whether any given rate vector is achievable when fanout-splitting is allowed, then this algorithm can be used to decide whether any given rate vector is achievable even without fanout-splitting. The intuition behind the argument is that, whenever a flow is served using fanout-splitting, it keeps the input busy for more time than if it had been sent out as one flow. So, in order to do fanout-splitting, the net inflow at the multicasting input should be low enough to accommodate this extra time requirement. If the net inflow adds up to exactly 1 packet per time slot, then even if fanout-splitting is allowed, it cannot be done.

Theorem 3 *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.*

Proof: 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. In this new case, fanout-splitting is allowed.

T' is achievable with fanout-splitting only if T is achievable without fanout-splitting. Even though fanout-splitting is allowed for T', the fact that the net inflow at the input is 1, implies that the fanout is never split during the schedule. Thus, any schedule for T' can be restricted to the flows in T, thus giving a schedule that serves T without splitting the fanout.

Suppose there is an easy algorithm to decide whether the rate vector of T' is achievable with fanout-splitting. Then the same answer holds for the question of whether the original rate vector of T is achievable without fanout-splitting.

□

Corollary 1 *The problem of deciding achievability in a multicast switch when fanout-splitting is allowed, is NP-hard.*

Proof: This follows from Theorem 2 and the reduction shown in Theorem 3 that any achievability deciding problem in the no-splitting case can be reduced to an instance of the achievability question in the fanout-splitting case. □

3.3 An Algorithm for Moderate Multicast

We now present an algorithm to decide the achievability of a given rate vector, with no fanout-splitting. The algorithm is shown to be fixed-parameter tractable in the number of multicasts, and is thus practical for a moderate multicast load. This algorithm naturally gives a schedule to achieve the rates in a stable manner.

Algorithm 1 DECIDER:

INPUT: A rate requirement vector $\mathbf{r}_o$; a traffic pattern in an $N \times N$ switch, with k multicasts and all possible unicasts.

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 achieve it in a stable manner.

1. *Let the multicasts be $M_1, M_2, \ldots, M_k$. Let $A \subseteq [k]$. 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. Compute R_A for all possible subsets A of $[k]$.*[1]

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 $\mathbf{B_j x} \leq \mathbf{c_j}, j = 1, 2, \ldots, J$ be the set of convex regions representing R_A. Verifying whether a point $\mathbf{r}$ is in the convex hull of these regions is equivalent to verifying whether the following linear program in the variables $\mathbf{y_j}$ and ϕ_j is feasible:

$$\sum_{j=1}^{J} \phi_j = 1; \quad \phi_j \geq 0; \quad \mathbf{r} = \sum_{j=1}^{J} \mathbf{y_j}; \quad \mathbf{B_j y_j} \leq \phi_j \mathbf{c_j} \quad \forall j = 1 \textit{ to } J. \tag{3.1}$$

Lemma 2 *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 subsets A of $[k]$.*

Proof:

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

Proof: Let $\mathbf{x}$ be any point in the convex hull. Then $\mathbf{x} = \sum_{i=1}^{2^k} \phi_i \mathbf{x_i}$, where $\mathbf{x_i} \in R_{A_i}$ and $0 \leq \phi_i \leq 1$; $\sum_i \phi_i = 1$. Since $\mathbf{x_i} \in R_{A_i}$, there is an offline schedule that achieves $\mathbf{x_i}$. Let this schedule be $(\mathbf{a}^{(i)}, \mathbf{S}^{(i)})$, 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, switch state is represented in terms of the incidence vector of the stable set corresponding to the configuration.) Hence,

$$\mathbf{x_i} \leq \sum_j a_j^{(i)} \mathbf{s_j^{(i)}} \tag{3.2}$$

Consider the schedule:
$([\phi_1 \mathbf{a}^{(1)}; \phi_2 \mathbf{a}^{(2)}; \ldots; \phi_{2^k} \mathbf{a}^{(2^k)}], [\mathbf{S}^{(1)}; \mathbf{S}^{(2)}; \ldots; \mathbf{S}^{(2^k)}])$, i.e. the new schedule is the concatenation of scaled versions of the old 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 (3.2).

[1] $[k]$ is a notation for the set {1,2,...,k}

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

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

$$\mathrm{x} \leq \sum_j a_j \mathrm{s_j} \tag{3.3}$$

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} subset of $[k]$. Let $\mathrm{S}^{(i)}$ be a subsequence of S such that the multicasts $\{M_j | j \in A_i\}$ are being served in all the switch states in $\mathrm{S}^{(i)}$. Let

$$\phi_i = \sum_{\{j | \mathrm{s_j} \in \mathrm{S}^{(i)}\}} a_j$$

$$\mathrm{x_i} = \frac{1}{\phi_i} \sum_{\{j | \mathrm{s_j} \in \mathrm{S}^{(i)}\}} a_j \mathrm{s_j}$$

This means $\mathrm{x_i}$ is a convex combination of states with the multicasts in A_i always connected. In other words, this means $\mathrm{x_i} \in R_{A_i}$. Also, substituting in inequality (3.3), $\mathrm{x} \leq \sum_i \phi_i \mathrm{x_i}$. This shows that x is in the convex hull of the R_A's. □

The next lemma characterizes the R_As explicitly, for any subset A of $[k]$.

Lemma 3 *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:*

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

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

Proof: 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 when fanout-splitting is not allowed. It also implies that if the multicasts do not overlap, they get a rate of exactly 1.

The 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 only the unicast inputs and outputs. This is precisely given by the two inequalities in the statement above (see [14]). $\square$

Corollary 2 *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 3, the rate region can be explicitly specified using at most one inequality for each input, output and multicast flow. If the number of multicast flows is polynomial in N, the switch size, the number of inequalities is thus polynomial in N. $\square$

Theorem 4 *Consider a traffic pattern in an $N \times N$ switch, with k 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 and exponential in k. Thus, deciding achievability is fixed parameter tractable in the number of multicasts.*

Proof: By Lemma 2, Algorithm 1 is correct.

We now have to show the complexity result. Clearly, the number of R_A's is 2^k (the number of subsets of $[k]$). By Corollary 2, 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 (3.1). Verifying feasibility of an LP can be done in time polynomial in the number of constraints and variables, using the ellipsoid algorithm [24]. Since the number of inequalities is polynomial in N and exponential in k, the theorem follows. $\square$

Schedule for Serving Achievable Rates

Note that the above algorithm in fact leads to an offline schedule for the achievable rate vectors. Once a feasible point is found for the LP given in equation (3.1), then the given rate vector can be expressed as a convex combination of vectors from different R_{A_i}'s. The sub-schedule that achieves each vector used in the convex combination can be found using the Birkhoff-von Neumann decomposition for unicast (given in [5] and [7]). These can then be concatenated as described in the proof of Claim 1 in Lemma 2.

3.4 Algorithm for Computing the Offline Schedule

In this section, we address the problem of computing the offline schedule in a general case. For perfect conflict graphs, even for a large number of multicasts, we get an efficient scheme.

Definition 3.4.1 (Fractional Weighted Coloring Problem) *The* fractional weighted coloring problem *is stated as follows:*
Minimize $\sum_{i=1}^{k} \lambda_i \quad (\lambda_i \in \mathbb{R}_+, \ \forall i)$ *such that there exist stable sets* $\{S_j\}$ *with* $\sum_{i=1}^{k} \lambda_i \chi^{S_i} = w$*, where* w *is a given weight vector.*

Theorem 5 *The problem of computing the decomposition of the given rate requirements into switch connection states (stable sets) is precisely an instance of the problem of* fractional weighted graph coloring *which is defined above. The weight vector* w *is chosen as the vector of required rates for the flows. The minimum of the optimization problem will be at most 1 for points within the rate region.*

Proof: The formal statement of the problem of decomposing the rate requirements into a sequence of valid switch configurations was given in Section 2.4, in terms of flows, their rates and switch connection states. Consider the inequality (2.3). Each flow f_i there, corresponds to a vertex here. The ϕ's in (2.3) correspond to the λ's here. For stable sets, χ is a binary vector. So, if we split the above vector equation into separate equations for each component of the vector, we get the same equation as in (2.3). When the minimum of the sum of λ's is less than or equal to 1, all the inequalities given in the problem statement are satisfied, and thus the point is within the rate region. $\square$

To find the scheduling algorithm, we assume the rates are all rational numbers. Multiplying by the LCM of the denominators converts the rates into integers. Thus we have one integer $n(v)$, say, assigned to each vertex v, proportional to the rate of that flow.

Suppose the graph is now colored in such a way that vertex v gets $n(v)$ colors, and no two adjacent vertices have any two colors in common. Then each color represents a stable set, and the coloring itself is the decomposition into stable sets, that we are looking for. This problem is known as the *weighted coloring problem* [14].

To perform the weighted coloring, each vertex v is "replicated" $n(v)$ times. The operation of replication is defined as follows: A vertex v is replicated by adding a new vertex v' which is adjacent (connected by an edge) to v and all its neighbors $N(v)$. Repeating this process n times on a vertex amounts to replacing the vertex by a clique of size n.

It can be seen that a normal coloring of the graph after replication has a one to one correspondence to a weighted coloring of the graph before replication. Besides, the process of replication preserves perfectness [14]. And coloring of perfect graphs can be done in polynomial time using the algorithm given in [16].

Thus, if the conflict graph is perfect, then it can be decomposed into stable sets and the offline schedule can be found in polynomial time.

It may be noted that, in the unicast case, weighted coloring of the conflict graph is equivalent to edge coloring of a bipartite multigraph. This approach to switch scheduling has been explored in the unicast case in [21]. Also note that, the coloring approach gives the BVN decomposition of the rate matrix using the smallest number of permutation matrices. The number of permutation matrices in the schedule has a direct impact on the cell delay guarantees.

3.5 Examples of Conflict Graphs – When is it perfect?

The characterization of the stable set polytope has been discussed in Section B. Several examples of conflict graphs can be found in [23] and they have also been described here.

For the case when the conflict graph is a perfect graph, the clique inequalities completely describe the set of all achievable rates. As explained in Section 3.4, for perfect graphs, the offline schedule can be found in polynomial time. Note that, in the unicast case, the conflict graph is the line graph of a bipartite graph, which is known to be perfect. Hence the clique inequalities are necessary and sufficient. The clique inequalities correspond to the admissibility conditions (hence the BVN rate region), since the only cliques are the set of all flows at the same input or at the same output.

Now, we characterize the rate region for some examples involving multicast in an $N \times N$ switch.

The interval graph case: When unicasts are not present, certain patterns of multicast flows lead to perfect conflict graphs. (This scenario is applicable in switches where there is a separate scheduler to handle the multicast flows.) One example of such a traffic pattern is the case of contiguous interval fanouts.

Corollary 3 *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 clique inequalities of the conflict graph suffice to specify the rate region.*

Proof: An interval graph is a graph where each vertex corresponds to an interval on the real line, and two vertices are connected iff the corresponding intervals have a non-empty intersection. For the traffic pattern given, the conflict graph is an *interval graph* which is known to be perfect [14]. Hence, from Theorem 1, the stable set polytope is the rate region and it is completely specified by the clique inequalities. □

The question is: what class of traffic patterns lead to perfect conflict graphs?

In general, there are several other traffic patterns which when modified in a suitable manner, yield a perfect conflict graph. Here is an example. Consider the pattern shown in Fig. 3-1. It consists of three multicast patterns, one each from inputs 1, 3 and 4, in a 5×5 switch. The fanouts do not form contiguous intervals, because the multicast from input 3, shown with a thick line, does not have a set of consecutive outputs (only outputs 2 and 4) as its fanout. But, if we add output 3 to its fanout deliberately, then we do not introduce any new conflict among the flows. So we might as well complete the fanout to include outputs

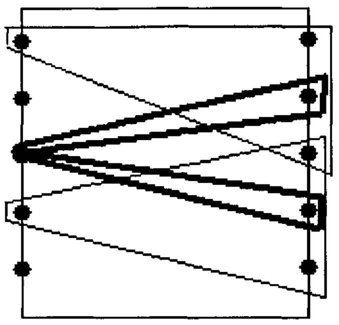


Figure 3-1: Example: The multicast from input 3 does not form a contiguous interval, but can be completed to one, without any new conflicts, thereby leading to an interval graph as the conflict graph

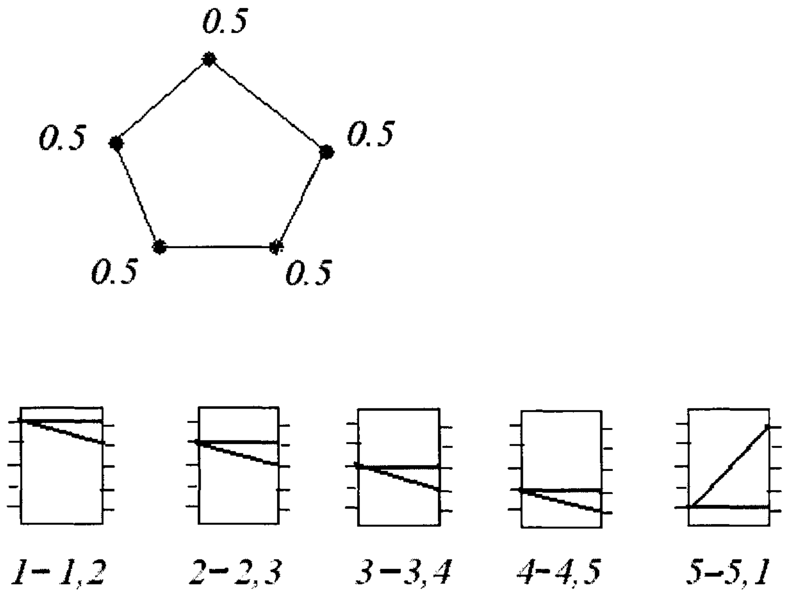


Figure 3-2: Example of case where clique inequalities do not suffice to characterize the rate region

2,3 and 4. Once this is done, the set of fanouts form contiguous intervals, and the conflict graph is again an interval graph, which is perfect.

Generalizing, *if it is possible to modify the given traffic pattern into an equivalent one, such that the conflict graph becomes perfect*, then the offline schedule can be found in polynomial time.

There are other options, if we are prepared to lose some optimality. The strong perfect graph theorem (see Appendix A) says that a graph is perfect if and only if it has no odd holes and no odd antiholes. We can make the conflict graph perfect by introducing conflicts (edges) that are actually not present, in order to break odd holes and odd antiholes. For instance, if we locate a hole, we can introduce chords to convert the odd hole into a set of even holes and a triangle. Alternately, we could choose the static splitting of multicasts in such a way that the conflict graph is perfect. These two ideas may lead to a smaller rate region than the actual one. But they can be used to design approximation heuristics to compute an offline schedule in polynomial time, especially when the traffic load is not close to capacity.

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 clash with any other flow. There is a complete connection between L and M – every vertex in L is connected to every vertex in M.

Corollary 4 *The rate region for the mixture of unicasts and broadcasts is completely specified by the clique inequalities of the conflict graph.*

Proof: From the strong perfect graph theorem, a graph is perfect if and only if it does not contain any odd hole or odd antihole as an induced subgraph. L is perfect (as it is the line graph of a bipartite graph), so it does not contain any odd hole or antihole. The same holds for M because it induces a complete graph, which is perfect. Thus, we need to check if there is any odd hole or antihole with some vertices in L and some vertices in M. Even this is not possible, because, in any odd hole or antihole, for every vertex, there are at least two vertices which are not its neighbors. But, if we choose any vertex from M, it will violate this condition, as it is connected to every vertex in the graph. Thus, the conflict graph is perfect. From Theorem 1, the rate region is given by the stable set polytope, which for a perfect graph is completely specified by the clique inequalities. $\square$

The maximal cliques are again of two kinds. One is the set of all unicast flows originating at the same input union the set of all the broadcasts. The other is the set of all unicast flows reaching the same output union the set of all the broadcasts. This leads to the following corollary:

Corollary 5 *The rate region for a traffic pattern consisting of a mixture of unicast and broadcast flows is given by:*

$$\sum_{i=1}^{N} r_{ij} + \sum_{i=1}^{N} r_{iB} \leq 1, \quad \forall j = 1 \dots N \tag{3.4}$$

$$\sum_{j=1}^{N} r_{ij} + \sum_{i=1}^{N} r_{iB} \leq 1, \quad \forall i = 1 \dots N \tag{3.5}$$

where r_{ij} is the unicast rate from input i to output j and r_{iB} is the rate of the broadcast from input i. $\square$

Note that this result agrees with intuition. It is as though we serve out the broadcasts first, and then serve the unicasts in the remaining time. So the rate region is the unicast region normalized to $(1 -$ sum of all broadcasts$)$. When this result is applied to the 2×2 case, we get the same region that as in Fig. 2-1. Note that even if some of the broadcasts are absent, the conflict graph is still perfect, because the induced subgraph of any perfect graph is perfect.

A non-perfect case: Fig. 3-2 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. More generally, if both unicast and multicast flows are present, then the conflict graph is in general not perfect, except when all the multicast flows are broadcast. This is because a multicast can 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 [14].

In summary, when both unicasts and multicasts are present, the conflict graph is generally not perfect. But there are traffic patterns that can be modified so that the conflict graph becomes perfect. Even otherwise, if the traffic load is relatively low, we can force the conflict graph into a perfect graph and then compute the offline schedule in polynomial time.

Chapter 4

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 was proposed for online unicast scheduling by McKeown in [18], to address these issues. A brief discussion of *i*-SLIP can be found in Section 2.3.1.

In this chapter, we propose one an online algorithm for multicast, based on *i*-SLIP, which utilizes the conflict graph idea to extend the well known *i*-SLIP unicast algorithm for multicast. This corresponds to the case when fanout-splitting is not allowed.

We compare it with the ESLIP algorithm which was suggested in [19] as an extension of *i*-SLIP for multicast. We use a slightly modified version of ESLIP for the comparison so that it works even for the case when there are multiple VOQs for multicast flows. This corresponds to 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 the 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 2.1.1.

4.1 Clique Algorithm

The motivation for this algorithm is the interpretation of i-SLIP in terms of the conflict graph formulation (Section 2.6). The i-SLIP algorithm is an iterative algorithm, with each iteration involving three steps, as described in Section 2.3.1. The key idea is to transfer the job of selecting grants from the outputs to cliques in an output conflict graph. Similarly, the job of accepting grants is now done by each clique in the input conflict graph. 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 clashes between flows only on the input side. Similarly, *output conflict graph* is the conflict graph obtained by considering clashes 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 the 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 one of its vertices — the vertex (flow) that 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. Therefore, 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 issue is resolved by allowing the loop to start with a different clique in different cell slots, in a round robin manner.

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.

4.2 A modification of ESLIP

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

Algorithm 3 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: 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. Unlike ESLIP, there are no separate common pointers for multicast flows. 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.

4.3 Performance Evaluation

Feasibility is not the only concern for a switching algorithm. It should also lead to low delay. Consider a rate point in the intersection of two possible rate regions, corresponding to two different choices of fanout splitting. Both choices will support the rate required. Since delay generally goes up as we approach the limit of the rate region, we expect that if we use a strategy in which the rate point is very close to the edge of the rate region, then we shall incur greater delay than if we use a strategy in which the rate point is far from the limits. To study this effect in greater detail, we have performed simulations of an 8×8 and a 16×16 multicast switch.

4.3.1 Simulation Setting — 16×16 switch

Two different traffic patterns were simulated. In both, besides all the unicasts, there is one broadcast flow from each input to all the outputs. Uniform i.i.d. Bernoulli traffic is used in the simulation.

The first traffic pattern has the following set of rates for the flows:

$$r_{i,j} = i/320, \quad \forall j, r_{i,B} = 0.005\alpha \text{ for } i = 1, \ldots, 13;$$
$$r_{14,B} = 0.018\alpha, r_{15,B} = 0.015\alpha, r_{16,B} = 0.012\alpha \quad (4.1)$$

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 variable used to control the load.

The second traffic pattern has the same unicast rates as the first one. The broadcast rates are however different:

$$r_{1,B} = 0.03\alpha, \quad r_{i,B} = 0.02\alpha \text{ for } i = 2 \text{ to } 7,$$
$$r_{8,B} = 0.01\alpha, \quad r_{i,B} = 0.005\alpha \text{ for } i = 9 \text{ to } 14,$$
$$r_{15,B} = r_{16,B} = 0.003\alpha. \quad (4.2)$$

Fig. 4-1 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 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 x-axis of the plot is the average of multicast fraction at all the inputs (i.e. the ratio of the 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.

The unicast rates have been chosen such that there is an uneven distribution of the unicast load on the inputs. This is done because, the effect of complete-copying being able to outperform no-fanout-splitting is observed well in such a scenario. When using i-SLIP for multicast, we have assumed that a copy network has performed complete fanout splitting of the multicast flows, and has fed the multiple copies into corresponding VOQs. The offline complete fanout splitting strategy corresponds indirectly to this online i-SLIP case. The clique algorithm does not do any fanout splitting. 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 complete fanout splitting 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 complete fanout splitting followed by i-SLIP.

The rate point for the second pattern is near the edge of the no-fanout-splitting capacity region, but is within the region corresponding to complete fanout splitting. 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 partial fanout splitting. Among online algorithms, those that allow partial splitting of the fanout are indeed expected to perform better, as there are fewer constraints in this case.

However, as mentioned earlier, in an offline algorithm, the choice of whether to split the fanout has to be made before hand using the rates and cannot be done in real time. In such cases, these simulation results indicate that there is no clear winner between the complete fanout splitting (copy) strategy and the partial or no splitting strategy.

4.3.2 8×8 Switch Simulation

A similar setting is used for the 8×8 switch simulation. The first traffic pattern used is:

$$r_{i,j} = i/80, \quad \forall j, \quad r_{i,B} = 0.01\alpha \text{ for } i = 1, \ldots, 5;$$
$$r_{6,B} = 0.049\alpha, \quad r_{7,B} = 0.037\alpha, \quad r_{8,B} = 0.024\alpha \quad (4.3)$$

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 variable used to control the load.

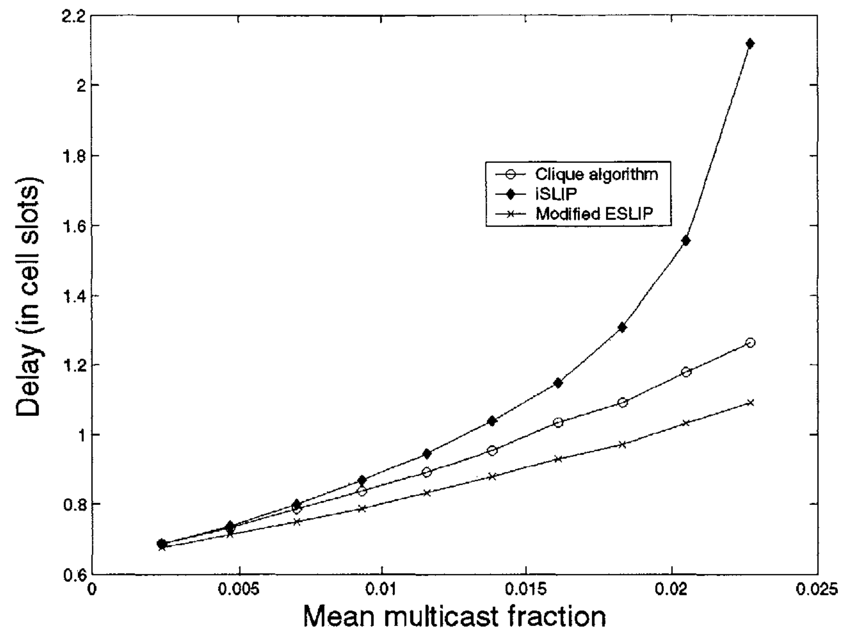
The second traffic pattern has the same unicast rates as the first one. The broadcast rates are however different:

$$r_{1,B} = r_{2,B} = 0.044\alpha, r_{i,B} = 0.02\alpha \text{ for } i = 3 \text{ to } 7, r_{8,B} = 0.01\alpha. \quad (4.4)$$

The plots are shown in Fig. 4-2 for the two traffic patterns. For the first pattern, since the rate point is near the boundary of the complete fanout splitting rate region, the copy based i-SLIP method gives a large delay compared to the other two algorithms. In the second case, the clique algorithm leads to a large delay because the rate point is now near the limit of the no fanout splitting rate region. The inferences are identical to the 16×16 case.

We can draw two main conclusions from the simulations. First, if dynamic fanout splitting is not allowed, then there are two options — use a copy network to convert the multicast into unicast flows, or else, directly multicast without any fanout splitting. The two traffic patterns considered here show that sometimes the copy method (i-SLIP) would work better while at other times, direct multicasting (clique algorithm) will do better. Thus, there is no single clear winner between the two, and one should choose carefully between the two options.

Second, in both cases, the modified ESLIP algorithm performs as well as the better of the two options discussed above. Thus if dynamic fanout splitting is allowed, then the modified ESLIP is a good option.



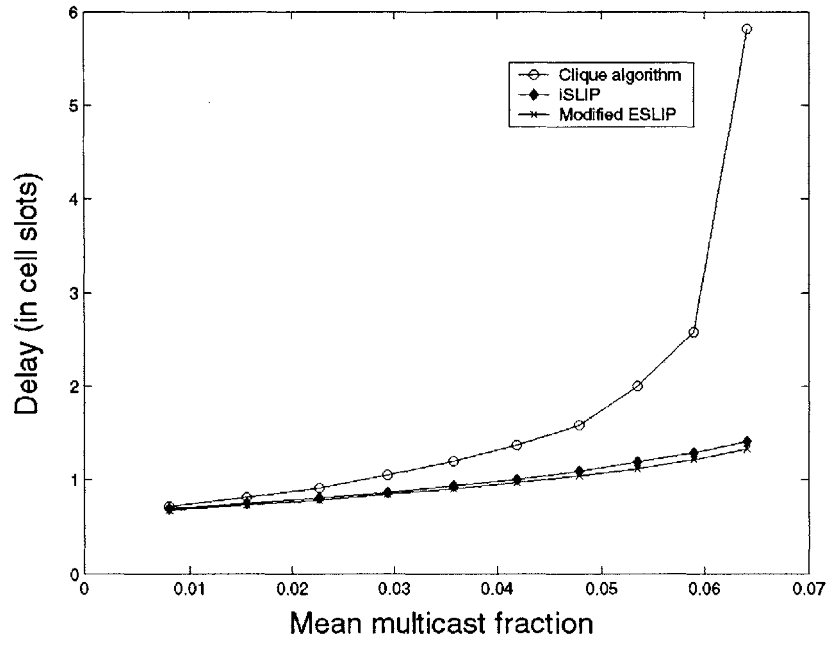
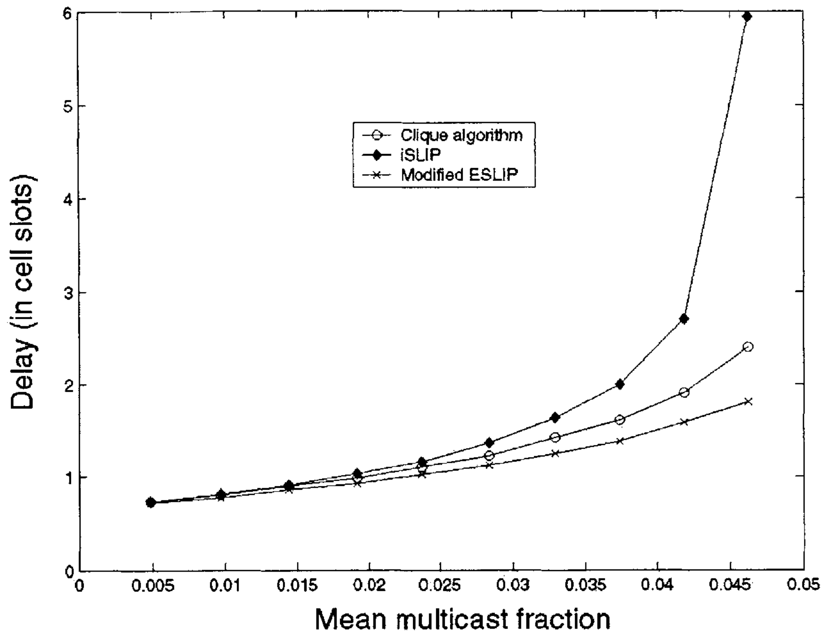


Figure 4-1: Delay versus multicast fraction: 16×16: cases 1 and 2 respectively. Simulation settings are given by Eqn.(4.1) and (4.2)



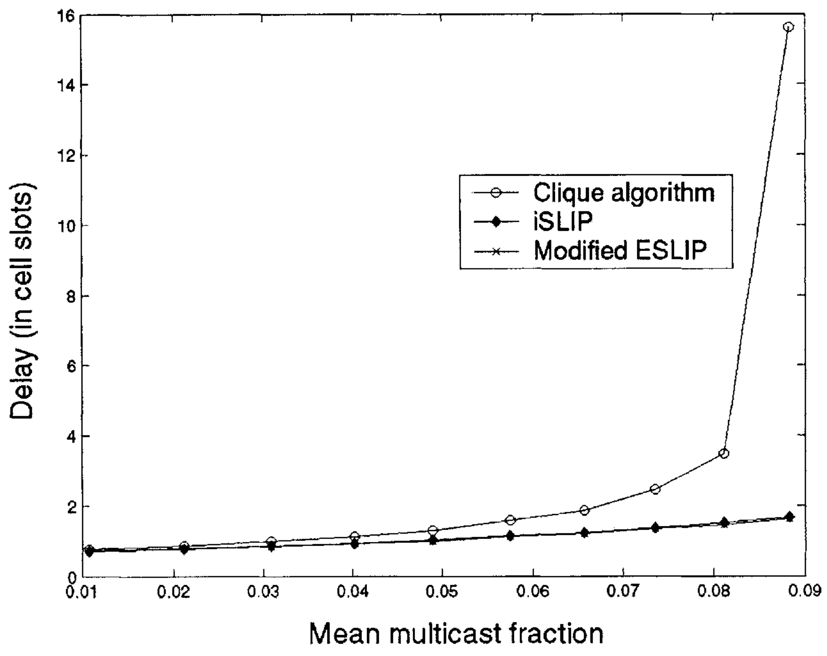


Figure 4-2: Delay versus multicast fraction: 8 $\times$ 8: cases 1 and 2 respectively. Simulation settings are given by Eqn.(4.3) and (4.4)

Chapter 5

Conclusions and Future Directions

This chapter summarizes the work presented in this thesis. In this work, we have explored the space of offline multicast switch scheduling, on the lines of the Birkhoff-von Neumann switch proposed for unicast by Chang et al. in [5]. The basic idea is that, if we have prior knowledge of the multicast flows as well as their average rates, then we can decompose the rate matrix into a convex combination of switch connection states which naturally gives an offline schedule.

Using a graph theoretic formulation, we have shown that the achievable rate region for the no-splitting case is the stable set polytope of the conflict graph. This leads to the result that deciding the achievability of, and finding a schedule for, a given set of multicast flows and rates, is NP-hard in general. However, we have shown that this can be done in polynomial time if the number of multicasts is of the order of the logarithm of the switch size. As the amount of multicast traffic is typically much less than the unicast traffic, this algorithm is useful in various scenarios. We have also shown that in general, finding the offline switch schedule is precisely the fractional weighted coloring problem. This naturally gives rise to a polynomial algorithm when the conflict graph is a perfect graph. We have given several examples of multicast flow patterns which lead to perfect conflict graphs.

In general, multiple approaches exist with respect to the type of fanout splitting used while serving the multicast. The static approach utilizes the knowledge of the multicast pattern and rates, and decides beforehand how exactly each flow will be split. This choice is kept fixed throughout the schedule. In the dynamic approach, the way the fanout is split is not kept fixed. The same flow may be split differently in different time slots. The decision on how split each flow in each time slot may be made in real time (online) or beforehand (offline).

The above stable set polytope result can also be used to compute the rate region for the static splitting approach, where the fanout splitting is decided offline, and kept fixed. We focus on static splitting because dynamic splitting requires an exponential number (in the switch size) of virtual output queues even if the multicast pattern is known beforehand, whereas static splitting only needs one queue per split flow.

We have shown through simulations that there is no clear winner between complete fanout-splitting and no-splitting in terms of rate region, if we exclude dynamic splitting. The rate region of dynamic splitting is expected to include the rate region of all other approaches, since it subsumes the other approaches. But, dynamic splitting presents implementation difficulties.

This result has implications in the design of optical networks. While implementing a multicast light tree in an optical network, the router can either split the light and do direct multicast, or it can make multiple copies in the network layer, followed by unicast in the physical layer. The results in this work indicate that neither of these strategies is a clear winner in terms of rate region and that one must choose between these approaches carefully.

In conclusion, this thesis presents a graph theoretic approach to multicast switch scheduling when certain restrictions are placed on the type of fanout splitting that can be used. The new formulation leads to a better understanding of the problem and enables us to use graph theoretic results to establish the rate region for multicast switching.

5.1 Future Work

The rate region cannot be explicitly characterized in the general case. However, the stable set polytope is known for several special classes of graphs other than perfect graphs. One possible line of future work is to study whether any of these results may be used to establish the rate region for some natural classes of traffic patterns.

Another important line of research is to find approximation algorithms for the various hard problems discussed in this work, such as computing the offline schedule.

In the dynamic splitting case, an easy characterization of the achievable rate region does not seem to be known except in the implicit manner given in [20]. It would be an interesting problem to characterize this region explicitly, or to provide with a graph theoretic formulation which may lead to a better understanding of the problem.

Appendix A

Graph-Theoretic Definitions

This appendix provides the definitions of several graph theory related terms used throughout this thesis. These definitions are based on [14].

Consider a graph $G(V, E)$, where V is the vertex set and E is the edge set. Let $w \in \mathbb{R}_+^{|V|}$ be the vector of weights assigned to the vertices.

Definition A.0.1 (Perfect matching) *A matching on G is a set of edges no two of which share a vertex in common. A* perfect matching, *if it exists, covers all the vertices in the graph.* □

The incidence vector of M, a matching 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 A.0.2 (Perfect matching polytope) *The* perfect matching polytope *of G is defined as:*

$$M(G) := conv\{\chi^M \in \mathbb{R}^{|E|} | M \subseteq E, M \text{ is a perfect matching of } G\} \tag{A.1}$$

□

It is the convex hull of the incidence vectors of all the perfect matchings of the graph.

Definition A.0.3 (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 S, a stable set 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 A.0.4 (Stable set polytope) *The* stable set polytope *of G denoted by $STAB(G)$ is defined as:*

$$STAB(G) := conv\{\chi^S \in \mathbb{R}^{|V|} | S \subseteq V, S \text{ is a stable set of } G\} \tag{A.2}$$

□

Definition A.0.5 (Clique) *A* clique *is a set of vertices in a graph, any two of which are adjacent to each other.* □

Definition A.0.6 (Odd hole) *A chordless cycle with an odd number of vertices (at least 5) is called an* odd hole. □

Definition A.0.7 (Odd antihole) *The graph complement of an odd hole is defined to be an* odd antihole. □

Definition A.0.8 (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.* □

Theorem 6 *(Strong perfect graph theorem) [14]. A graph is perfect if and only if it contains no odd hole and no odd antihole.* □

Definition A.0.9 (Hypergraph) *A graph in which edges may connect two or more than two vertices is called a* hypergraph. □

Definition A.0.10 (Hyperedge) *An edge of a hypergraph which connects two or more vertices is called a* hyperedge.

Appendix B

Characterizing the Stable Set Polytope

The explicit characterization for the stable set polytope of a general graph is not yet known. Since finding a maximum stable set is NP-complete, one does not expect to find a polynomial-time checkable system of linear inequalities describing the stable set polytope of a general graph. However, there are several classes of graphs for which the polytope can be explicitly characterized. This appendix outlines some of the results known in this direction. The source of the results presented in this chapter is [14]. The definitions of terms used in this chapter can be found in the Appendix A.

Like any other polytope, the stable set polytope has a representation as the intersection of half-spaces, but the complete set of facet-defining inequalities is unknown for most graphs. However, there are many valid inequalities known. If $\mathbf{x}$ is a point inside the stable set polytope of a graph $G = (V, E)$, then the following hold:

1. Non-negativity constraints:

$$0 \leq x_i \leq 1 \text{ for } i \in V$$

2. Edge inequalities:

$$x_i + x_j \leq 1 \text{ for } ij \in E$$

3. Odd circuit inequalities:

$$\sum_{i \in C} x_i \leq \frac{|C|-1}{2} \text{ for every odd circuit } C$$

4. Clique inequalities:

$$\sum_{i \in Q} x_i \leq 1 \text{ for every clique } Q$$

5. Rank constraints:

$$\sum_{i \in G'} x_i \leq \alpha(G')$$

   where α denotes the stable set number and G' is any subgraph of G.

In some cases, these constraints are necessary and sufficient conditions:

- Constraints 1 and 2 completely determine the stable set polytope, if and only if, G is a *bipartite* graph.
- Constraints 1, 2 and 3 completely determine the stable set polytope, if and only if, G is a *t-perfect* graph.
- Constraints 1 and 4 completely determine the stable set polytope, if and only if, G is a *perfect* graph.
- Constraints 1, 3 and 4 completely determine the stable set polytope, if and only if, G is a *h-perfect* graph.
- Constraints 1 and 5 completely determine the stable set polytope, if and only if, G is a *rank-perfect* graph.

Bibliography

[1] G. Birkhoff, "Tres observaciones sobre el algebra lineal," *Univ. Nac. Tucuman Rev. Ser. A*, Vol. 5, pp. 147-151, 1946.

[2] 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.

[3] N. McKeown, V. Anantharam, J. Walrand, "Achieving 100% Throughput in an Input-Queued Switch", *Proceedings of IEEE INFOCOM 1996*, pp. 296-302.

[4] T.T. Lee, "Nonblocking copy networks for multicast packet switching", *IEEE Journal on Selected Areas in Communications*, Volume: 6 , Issue: 9 , Dec. 1988 pp. 1455 - 1467

[5] C.S. Chang, W.J. Chen and H.Y. Huang, "Birkhoff-von Neumann input buffered crossbar switches," *Proceedings of IEEE INFOCOM 2000*, pp. 1614-1623, Tel-Aviv, Israel, March 2000.

[6] 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," *Proc. of 7th IEEE International Workshop on Quality of Service (IWQoS)*, pp. 79-86, London, 1999.

[7] C.S. Chang, W.J. Chen and H.Y. Huang, "Birkhoff-von Neumann input buffered crossbar switches for Guaranteed-Rate Services," *IEEE Transactions on Communications*, Vol. 49, pp.1145-1147,July 2001.

[8] M. Andrews, S. Khanna, K. Kumaran, "Integrated Scheduling of Unicast and Multicast Traffic in an Input-Queued Switch", *Proceedings of IEEE INFOCOM 1999* The Conference on Computer Communications, no. 1, March 1999 pp. 1144-1151.

[9] J. Li, N. Ansari, "Enhanced Birkhoff-von Neumann Decomposition Algorithm for Input Queued Switches," *IEE Proceedings in Communications*, Vol. 148, No. 6, pp. 339-342, 2001.

[10] C.S. Chang, D.S. Lee, Y.S. Jou, "Load balanced Birkhoff-von Neumann Switches", *IEEE Workshop on High Performance Switching and Routing*, 2001.

[11] Jinhui Li, "Scheduling Algorithms for High Speed Switches", Ph.D. Dissertation, May 2001. (available online: The New Jersey Institute of Technology's electronic Theses & Dissertations)

[12] A.K. Parekh, R.G. Gallager, "A generalized processor sharing approach to flow control in integrated service networks: the single node case," *IEEE/ACM Transactions on Networking*, Vol. 1, Issue 3 pp. 344-357, June 1993.

[13] A.K. Parekh, R.G. Gallager, "A generalized processor sharing approach to flow control in integrated service networks: the multiple node case," *IEEE/ACM Transactions on Networking*, Vol. 2, Issue 2 pp. 137-150, April 1994.

[14] A. Schrijver, "Combinatorial Optimization: Polyhedra and Efficiency", Springer Verlag, 2003.

[15] M. Grötschel, L. Lovász, A. Schrijver. "Geometric algorithms and combinatorial optimization", Second edition. Springer-Verlag, Berlin, 1993.

[16] M. Grötschel, L. Lovász, A. Schrijver. "Polynomial algorithms for perfect graphs" in "Topics on perfect graphs". North-Holland Mathematics Studies, 88 (C. Berge and V. Chvtal, eds.), *Annals of Discrete Mathematics*, 21. North-Holland Publishing Co., Amsterdam-New York, 1984. xiv+369 pp. ISBN: 0-444-86587-X

[17] B. Prabhakar, N. McKeown, R. Ahuja, "Multicast scheduling for input queued switches", *IEEE Journal on Selected Areas in Communications*, Vol. 15, Issue 5, June 1997, pp. 855-866.

[18] N. McKeown, "The i-SLIP scheduling algorithm for input-queued switches", *IEEE/ACM Transactions on Networking*, Vol. 7, no. 2, Apr. 1999.

[19] N. McKeown, "A Fast Switched Backplane for a Gigabit Switched Router", *Business Comm. Review*, Vol. 27, No. 12, Dec. 1997

[20] M. A. Marsan, A. Bianco, P. Giaccone, E. Leonardi, F. Neri: "Multicast traffic in input-queued switches: optimal scheduling and maximum throughput." *IEEE/ACM Trans. on Networking* Vol. 11, No. 3, June 2003, pp. 465-477.

[21] G. Aggarwal, R. Motwani, D. Shah, A. Zhu: "Switch Scheduling via Randomized Edge Coloring", *44th Annual IEEE Symposium on Foundations of Computer Science* (FOCS'03), 2003, pp. 502.

[22] R. M. Karp, "Reducibility among combinatorial problems", *Complexity of computer computations* (Plenum Press, New York, 1972)

[23] J. Sundararajan, S. Deb, M. Médard, "To copy or not to copy: Extending the Birkhoff-von Neumann switching strategy to multicast switches", *LIDS Publication* # 2624, MIT.

[24] Khachiyan, L.G., A polynomial algorithm in linear programming, *Soviet Mathematics Doklady* 20 (1979), pp. 191-194.

[25] N. McKeown, "Optics inside Routers", *Proceedings of ECOC 2003*, Rimini, Italy, Vol. 2, pp.168-171, 2003.

[26] N. Singhal, B. Mukherjee, "Architectures and Algorithms for Multicasting in Optical WDM Mesh Networks using Opaque and Transparent Optical Cross-connects," *Proceedings, IEEE OFC 2001*, Anaheim, CA, March 2001.

[27] A. Zsigri, A. Guitton, and M. Molnar, "Construction of light-trees for WDM multicasting under splitting capability constraints," *10th International Conference on Telecommunications* IEEE, New York, 2003, pp. 171-175.