UNIVERSITY OF CALIFORNIA,

IRVINE

A THEORY OF LEXICOGRAPHIC OPTIMIZATION FOR COMPUTER NETWORKS

DISSERTATION

Submitted in partial satisfaction of the requirements

for the degree of

DOCTOR OF PHILOSOPHY

in Electrical and Computer Engineering

By

Jordi Ros Giralt

Dissertation Committee:
Professor Wei K. Tsai, Chair
Professor Scott Jordan
Professor Glenn Healey
Professor R. Jayakrishnan
Professor Ender Ayanoglu

2003
The dissertation of Jordi Ros Giralt is approved and is acceptable in quality and form for publication on microfilm:

________________________________
________________________________
________________________________
________________________________
________________________________

Committee Chair

University of California, Irvine

2003
TO LAURA
# TABLE OF CONTENTS

*LIST OF FIGURES*  
xii  

*LIST OF TABLES*  
xvi  

*LIST OF SYMBOLS*  
xviii  

*ACKNOWLEDGEMENTS*  
xx  

*ABSTRACT OF THE DISSERTATION*  
xxii  

*CHAPTER I*  
1  

*INTRODUCTION*  
1  

1 *From Babel Towers to Network Protocols*  
1  

1.1 *Contemporary Constraints and Protocols*  
3  

1.2 *Mapping Constraints and Protocols: Circuit Switching or Packet Switching*  
5  

2 *Network Assumptions in this Dissertation*  
9  

3 *Problem Statement*  
11  

4 *About this Dissertation*  
11  

*CHAPTER II*  
12  

*LEXICOGRAPHIC OPTIMIZATION: EXISTENCE, GLOBALITY AND UNIQUENESS*  
13
<table>
<thead>
<tr>
<th>Chapter</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Introduction</td>
<td>13</td>
</tr>
<tr>
<td>2</td>
<td>Preference (POP) and Lexicographic (LOP) Optimization Problems</td>
<td>14</td>
</tr>
<tr>
<td>3</td>
<td>Existence in a LOP</td>
<td>19</td>
</tr>
<tr>
<td>4</td>
<td>Globality in a LOP</td>
<td>19</td>
</tr>
<tr>
<td>5</td>
<td>Uniqueness in a LOP</td>
<td>23</td>
</tr>
<tr>
<td>6</td>
<td>POPs, Ordinal Versus Cardinal Optimization</td>
<td>24</td>
</tr>
<tr>
<td>7</td>
<td>Network Optimization and the Maxmin Problem</td>
<td>25</td>
</tr>
</tbody>
</table>

**CHAPTER III**

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Introduction</td>
<td>29</td>
</tr>
<tr>
<td>2</td>
<td>The Constrained Maxmin Problem</td>
<td>30</td>
</tr>
<tr>
<td>3</td>
<td>Extended Network</td>
<td>32</td>
</tr>
<tr>
<td>4</td>
<td>Maxmin Theory</td>
<td>33</td>
</tr>
<tr>
<td>5</td>
<td>CPG Algorithm</td>
<td>39</td>
</tr>
<tr>
<td>5.1</td>
<td>Single-link Algorithm</td>
<td>40</td>
</tr>
<tr>
<td>5.2</td>
<td>Multi-link Algorithm</td>
<td>46</td>
</tr>
<tr>
<td>6</td>
<td>Theory of Maxmin Bottleneck Ordering</td>
<td>50</td>
</tr>
<tr>
<td>6.1</td>
<td>Bottleneck Maxmin Order Definition</td>
<td>51</td>
</tr>
<tr>
<td>6.2</td>
<td>Precedent Link Relationship</td>
<td>52</td>
</tr>
</tbody>
</table>
6.3 CPG Graph

6.4 CPG Algorithm

CHAPTER IV

DISTRIBUTED UNICAST MAXMIN PROTOCOL

1 Introduction

2 Stateless or State Maintained Protocols

3 Distributed Protocol

3.1 Signaling Protocol

3.2 Rate Computation Algorithm

3.3 Protocol Convergence

3.4 Fluid Model: the Maxmin Vessel

4 Simulations

4.1 Simulation Setup

4.2 Response Time

4.3 Dynamic Convergence

CHAPTER V

MULTICAST MAXMIN THEORY

1 Introduction

2 Multicast Maxmin Theory
# LIST OF FIGURES

## CHAPTER I

- Figure 1 The waggle dance. The frequency of each waggle dance cycle indicates the distance; the angle $\alpha$ indicates the relative location of the food with the sun. ................................................................ 2
- Figure 2 Communication operations: propagation and forward. .......................................................... 5
- Figure 3 Circuit switching (left) and packet switching (right). ................................................................. 6

## CHAPTER II

- Figure 1 Partial order and total order examples ...................................................................................... 15
- Figure 2 POP examples ............................................................................................................................ 18
- Figure 3 Network optimization problem .................................................................................................. 25
- Figure 4 Maxmin problem ....................................................................................................................... 26

## CHAPTER III

- Figure 1 Network example with minimal and maximal rate constraints .............................................. 32
- Figure 2 Extended network transformation ............................................................................................. 33
- Figure 3 Single link projection algorithm (see Matlab function RATEsinglelink() in Appendix) ... 40
- Figure 4 Single-link network example .................................................................................................... 43
- Figure 5 Convergence of SolveSingleLinkCMM to the maxmin solution ........................................... 43
- Figure 6 SolveNetworkCMM algorithm (see Matlab function CPG() in Appendix) .. 47
- Figure 7 Extended network for network in Figure 1 .............................................................................. 49
- Figure 8 Precedence link relationship .................................................................................................... 54
- Figure 9 Network example ....................................................................................................................... 55
- Figure 10 CPG graph ............................................................................................................................... 56
- Figure 11 BuildCPGGraph algorithm ................................................................................................... 59

## CHAPTER IV

- Figure 1 Source algorithm ......................................................................................................................... 65
- Figure 2 Destination algorithm .................................................................................................................. 66
- Figure 3 Flow architecture of the multicast switch .................................................................................. 66
- Figure 4 Switch algorithm ......................................................................................................................... 67
- Figure 5 Network example ....................................................................................................................... 69
Figure 6 A wrong ER value of 15 is advertised, the correct ER value should be 10..... 70
Figure 7 Network transformation................................................................. 71
Figure 8 ComputeAR Procedure to solve the single-link case ....................... 72
Figure 9 Convergence steps......................................................................... 77
Figure 10 The Maxmin Vessel........................................................................ 79
Figure 11 BuildVessel algorithm.................................................................... 81
Figure 12 FindWaterLevel algorithm ............................................................ 81
Figure 13 ComputeAR using the fluid model approach................................. 81
Figure 14 UpdateVessel algorithm ................................................................ 83
Figure 15 ComputeAR using the fluid model with an update approach.......... 84
Figure 16 Prototype 1.0 of the lexicographic vessel...................................... 84
Figure 17 Network configuration.................................................................. 85
Figure 18 Response time of (a) our distributed algorithm (b) ERICA .......... 86
Figure 19 Dynamic rate allocation under high priority background traffic...... 88
Figure 20 Queue sizes at switches 1 and 2 .................................................... 88

CHAPTER V
Figure 1 Multicast communication................................................................. 90
Figure 2 Definitions in a multicast VC_i ....................................................... 94
Figure 3 Single link projection algorithm....................................................... 104
Figure 4 SolveNetworkCMMM algorithm...................................................... 105
Figure 5 Network configuration..................................................................... 108
Figure 6 Network configuration................................................................... 111
Figure 7 CPG iterations to solve the network in Figure 6 ......................... 112
Figure 8 Source algorithm ......................................................................... 114
Figure 9 Destination algorithm ................................................................... 115
Figure 10 Flow architecture of the multicast switch..................................... 116
Figure 11 Switch algorithm ......................................................................... 117
Figure 12 Solving the switch rate allocation problem................................. 120
Figure 13 ComputeAR Procedure to solve the single-link case ................. 121
Figure 14 Network configuration................................................................. 124
Figure 15 Transmission rates for each of the low priority flows.................. 125
Figure 16 Queue size at switch sw3 for traffic forwarded to sw1 ............... 126
CHAPTER VI

Figure 1 Optical switch architecture................................................................. 131
Figure 2 Example of Maxmin Wavelength Assignment..................................... 133
Figure 3 The classic maxmin definition cannot be applied to the discrete case........ 135
Figure 4 In this network, not all the discrete maxmin solutions are lexicographic optimal ................................................................. 140
Figure 5 d-CPG procedure (see Matlab function CPG() in Appendix) .............. 142
Figure 6 ComputeRatesHeuristicGeneric procedure ........................................ 143
Figure 7 Network example for d-CPG algorithm ............................................... 144
Figure 8 ComputeRatesHeuristicLEX procedure ............................................... 148
Figure 9 Experimental set-up............................................................................. 149
Figure 10 Only 4 counterexamples are found whose distance to the lexicographic optimal solution is more than 1 among the set of more than 1 million tested networks with $\|\Gamma\| + |\Omega| \leq 10$ .................................................................................... 151
LIST OF TABLES

CHAPTER I
Table 1 mapping protocols and constraints ................................................................. 7

CHAPTER IV
Table 1 Convergence times ...................................................................................... 77
Table 2 Computation of a new vessel upon the arrival of a new feedback $B_i^{new}$ .......... 83

CHAPTER V
Table 1 CPG algorithm execution .......................................................................... 109
Table 2 Bottleneck optimality condition check ....................................................... 110
Table 3 Maxmin rate solution ............................................................................... 124

CHAPTER VI
Table 1 Number of counterexamples for networks with $|\Gamma| + |\Omega| \leq 10$ and with link capacities $\{10, 15, 20, ..., 5 |\Gamma| + 5\}$ ............................................................................................................. 150
LIST OF SYMBOLS

- $\gamma_i$: the \textit{ith} element of a vector $\gamma$.

- $d[i, j]$: the element located in row $i$ column $j$ in matrix $d$.

- $d[:;k]$: the vector column defined by column $k$ of matrix $d$.

- $d[j,:]$ : the vector row defined by row $j$ of matrix $d$.

- $|\Phi|$: cardinality of a set $\Phi$.

- $\mathbb{R}$: the vector space formed from the real numbers.

- $\mathbb{Z}$: the ring formed from the integer numbers.

- $S^n$: an $n$-dimensional set with elements of the form $[s_1, s_2, ..., s_n]$, with $s_i \in S$.

- $S_+$: set that includes all the positive non-zero numbers in $S$, e.g. $\mathbb{Z}_+$.

- Rate vector $r$: a vector in $\mathbb{R}^m$, for some $m$, such that $r_i$ corresponds to the rate of flow $i$.

- Rate set $r$: a set that contains all the possible rates in a rate vector $r$.

- Rang of a matrix $\Gamma$, $\text{RANG}([\Gamma])$: is the smallest dimension of the matrix obtained with the linearly independent rows and columns of matrix $\Gamma$.

- $\Omega \rightarrow_c \text{maximize}$: the operation of finding the largest element in the set $\Omega$ in the $C$-sense.

- $\|x\|_1 = \sum_i |x_i|$: the $\ell - 1$ norm, in occasions used to measure the distance between two vectors.
ACKNOWLEDGEMENTS

It may sound familiar for an acknowledgement section, but it is simply a reality, that this work is the result of a distributed effort. Not only from those that have shared with me research discussions at the ET building. But also, from those that have not physically been close to me and continued giving their firm support.

My most sincere gratitude to Professor Wei Kang Tsai. He has taught me many things about the places he has been, and he has taught me to enjoy research.

I am very grateful to Pete Balsells, for providing the fellowships that let students like me come to this country in a program jointly funded with Generalitat de Catalunya, the government from Catalonia, my country. Also, to University of California, Irvine, for continuing Mr. Balsells’ support through additional scholarships.

It has been an additional effort to be so far away from my hometown best friends, but it is rewarding when they let me know that true friendship knows no distance. Some of them have showed me how far it can reach.

I also thank my best friends in California, Calvin, Wendy, Koji, Kevin, Henry and Mehran, from our startup Irvine Networks, later Xiran, and Oum. Special thanks to Calvin, for showing me the art of coding and, more importantly, for hosting me in his enjoyable oriental world.

Finally, I want to thank my family. To Cesca for always bringing her piano music to our ears when I was at home, to Lluis for always being the one reference for me. And special thanks to my mom and my dad, for being always there.
CURRICULUM VITAE

Jordi Ros Giralt

1999-2003: PhD in Electrical and Computer Science, University of California, Irvine.
2002-2005: Master in Business Administration (in progress), University of California, Los Angeles.

Publications and dissertations

“Maxmin Lambda Allocation for DWDM Networks”, IEEE ICON.
“A Theory of Convergence Order of Maxmin Rate Allocation and an Optimal Protocol”, IEEE INFOCOM.
“A General Theory of Constrained Max-Min Rate Allocation for Multicast Networks”, IEEE ICON (Best Student Paper Award).
“Bottleneck Branch Marking for Noise Consolidation in Multicast Networks”, IEEE ICON.
“Minimum Rate Guarantee without Per-Flow Information”, IEEE ICNP.
ABSTRACT OF THE DISSERTATION

A THEORY OF LEXICOGRAPHIC OPTIMIZATION FOR COMPUTER NETWORKS

By

Jordi Ros Giralt

Doctor of Philosophy in Electrical and Computer Engineering

University of California, Irvine, 2003

Professor Wei K Tsai, Chair

This dissertation presents a theory of network optimization based on the lexicographic criterion. The objective is to identify the transmission rates of each user that optimally utilize the network resources. Optimality is defined in two senses: on one hand, transmission rates are to be maximized; on the other hand, fairness among all the users must be guaranteed. The lexicographically largest point is the preferred one since, by definition, such solution recursively maximizes the rate of those users that are most poorly treated. The theorems of bottleneck optimality condition and projection optimality condition are derived, which allows the design of practical distributed protocols in three different scenarios: unicast, multicast and discrete rate communications. While the complexity of these protocols is minimal (logarithmic), their convergence time is shown to be two times faster than previous works. The theorem of precedent link is presented. This results into a theory of bottleneck order, which we use in two ways: to unveil the bottleneck structure of a network and to measure the complexity of a protocol.
Chapter I

INTRODUCTION

“Humilitat és caminar en la veritat”

Mare Teresa de Calcuta

1 From Babel Towers to Network Protocols

When a scout worker bee discovers food, she returns to her hive. Shortly after her return, many foragers bees leave the hive and fly directly for their meal. If the food is located at distances shorter than 50 meters, the scout worker will dance in circles the round dance. For distances longer than 50 meters, on a vertical surface of the hive she will dance a more elaborated tail-wagging dance resembling the shape of an eight. Such dances communicate information regarding the position of the food to the foragers that surround and watch carefully the scout’s motions before their departure. The Information is conveyed by the means of polar coordinates (Figure 1). The speed of the dance indicates the distance (radial coordinate), the faster the bee waggles, the closer the food is located. The angle between the straight-line portion of the tail-waggling dance and the
vertical with respect to earth indicates the relative position of the hive-food vector and the hive-sun vector when the sun is projected on the plane of the earth [FRI73].

Figure 1 The waggle dance. The frequency of each waggle dance cycle indicates the distance; the angle $\alpha$ indicates the relative location of the food with the sun

Ants secrete hormones on the floor to record the path back to a sugar cube. Bats use ultrasonic sounds and signal correlation methods to detect their preys in the dark. Men and women communicated first using manual gestures. Later, improvements of the human vocal system allowed for spoken language, which provided a way to communicate in the dark as well as free their hands when hunting.

Such are the countless communication methods that species have developed throughout 3.5 billion years of life’s history. They all have two common characteristics: for any
communication system, there exists a set of physical limitations; on those limitations, there must exist a communication method that brings the best performance. So for instance, the bees’ communication system is limited in the sense that only visual gestures can be utilized during the daylight. Thus, a dance becomes their best communication method.

We refer to physical limitations and communication methods as constraints and protocols, respectively. Today the quantity and variety of communication methods is vast. Yet, four major developments have influenced their limitations for the last two centuries: the discovery of the electromagnetism, the invention of the transistor, the development of information theory and the invention of fiber optics.

The previous four developments have modified the physical limitations of our communication systems (constraints). Just as our ancestors adopted a new spoken language (protocol) after the enhancement of the human vocal system (constraint), upon the above changes, we have had to invent new communication methods (protocols).

1.1 Contemporary Constraints and Protocols

The foundations of communications technology lay in the discovery of the electromagnetism by Oersted in 1820. Ampere, Faraday and Henry, extensively contributed to the development of the theory during the XIX century. In 1876, Alexander Graham Bell developed the first viable telephone system. By 1892, his company, American Telephone and Telegraph Company (AT&T), had developed a network of
240,000 telephone instruments\textsuperscript{1}. In 1947, Bardeen, Shockley and Brattain, all at Bell Labs, invented the transistor. This invention, together with the discovery of information theory by Shannon (1948), opened the door to the digital world. In 1966, Kao and Hockham's published in the July Proceedings of the Institution of Electrical and Electronics Engineers a scientific based forecast that fiber loss could be reduced below 20 dB/km. In 1970, Corning Inc. announced they had made single-mode fibers with attenuation below 20 dB/km. A viable fiber optic technology was then invented that opened the door to high-speed communications.

All the above discoveries have redefined the limitations of our communication systems. If we were to interpret this change in terms of an optimization problem, we would say that the constraints of our objective set have been modified. For each set of constraints, there must exist an optimal protocol that provides the best communication performance, based on a particular criterion. Thus, researchers have had to deal with the problem of mapping the most suitable protocol to the given constraints. Throughout the last century, two major protocols have been invented: circuit switching and packet switching. Just like the bees, such protocols specify the way humans should waggle dance in a constrained world.

\textsuperscript{1} At its early stage, the telephone network grew faster than the Internet
1.2 Mapping Constraints and Protocols: Circuit Switching or Packet Switching

In order to convey information from one location to another, two major operations are required: propagation and forwarding (Figure 2). To illustrate this concept, let us consider the postal service. The sender drops a parcel at the post office, which performs the first forwarding operation and hands it to a truck. Then, the truck propagates the parcel to the next post office. The operation of forwarding at each hop (post offices) and propagating the parcel (by the means of trucks, aircrafts or boats) constitute the postal office communication protocol.

![Figure 2 communication operations: propagation and forward](image)

Let $\gamma$ be the set of propagation and forwarding elements in a communication system. For instance, $\gamma$ may include trucks, aircrafts, postal offices, telephones, computers, copper cables or fiber optics, among many others. We use $\pi$ and $\phi$ to denote the amount of propagation and forwarding resources of $\gamma$, respectively. Both $\pi$ and $\phi$ are measured in terms of amount of information transmitted per unit of time. Since it is a function of the current technology limitations, $\gamma$ defines the constraints. Now given a $\gamma$, the design of a protocol that optimally utilizes the constraints will depend on the values of $\pi$ and $\phi$. In particular, if $\pi > \phi$ then an optimal protocol should try to minimize the overheads of
the forwarding operations. Likewise, if $\varphi > \pi$ then one should try to minimize the overheads of the propagation operations. Circuit switching and packet switching are two types of protocols that are designed according to this relation.

In circuit switching, a set up phase starts a transaction by configuring all the forwarding elements to establish a fixed path between a source and a destination (left picture in Figure 3). Once this phase is completed, the path becomes fully dedicated to this source-destination pair and the actual information can now be transmitted. Upon termination, a tear down phase is performed that releases the path so that its forwarding elements can be used by other source-destination pairs.

![Figure 3 Circuit switching (left) and packet switching (right)](image)

On the other hand, a packet switching protocol transmits information as soon as it is available. Upon reception of data, a forwarding element has to decide what the next forwarding element is and transmit such data towards that element (right picture in Figure 3). Because there is no path reservation, forwarding elements can virtually operate
several source-destination pairs at the same time. Therefore, in packet switching there is no setup / tear down phases (Table 1).

<table>
<thead>
<tr>
<th>Constraint condition</th>
<th>Preferable protocol</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \pi &gt; \phi )</td>
<td>Circuit switching</td>
</tr>
<tr>
<td>( \phi &gt; \pi )</td>
<td>Packet switching</td>
</tr>
</tbody>
</table>

*Table 1 mapping protocols and constraints*

By the time the telephone network was born, the transistor had yet to be invented. Thus, there was no viable technology to store and forward voice information in a real time basis. In other words, the information would have to be sent from source straight to destination because the forwarding resources where very limited or, using our notation, because \( \phi = 0 \). On the other hand, the first propagation resources that were used to develop the telephone network where made of copper. While their capacity was very limited, the invention of electromagnetism led us to the constraint \( \pi > 0 \), which meant \( \pi > \phi \). Therefore, the most suitable protocol became circuit switching and such was the design of the Plain Old Telephone Service, the original telephone network.

When in 1947 the transistor was invented, the \( \phi \)-constraint became relaxed to \( \phi > 0 \). Since then, the forwarding resource has experienced a growth rate known as the Moore’s Law: the number of transistors per square inch in integrated circuits (i.e. the value of \( \phi \)) doubles approximately every 18 months. In 1961, Leonard Kleinrock published the first
paper in packet switching theory\textsuperscript{2}. In 1969, the $\varphi$-resource crossed over the $\pi$-resource reversing the constraints to $\varphi > \pi$. Therefore, based on the new state of art in technology, Kleinrock’s idea became economically viable. That same year, the first Internet message using packet switching technology was sent from University of California, Los Angeles, to Stanford Research Institute.

Today the telecom industry, which originally based their networks on circuit switching technology, is converging to a world where switches forward packets. Telephone, television, video streaming, video on demand, email, world wide web, teleconferences... they are all trending towards the packet switching technology. However, history has shown us that to understand the longer-term trends, we need to look at the constraints. The discovery of fiber optics has brought the last change in the constraints domain. The capacity of the communication lines, the propagation resource, has seen a notable increase due to this discovery. Since 1995 the $\pi$ resource has doubled every 7 months. Because this rate is higher than the rate dictated in Moore’s Law, one would expect a new reverse effect, $\pi > \varphi$.

Whether a new transition from packet switching to circuit switching will happen is uncertain. While this transition can be justified from a mathematical standpoint, legacy or regulation issues will play a decisive role to the design of the future Internet.

\textsuperscript{2} Leonard Kleinrock, "Information Flow in Large Communication Nets.", RLE Quarterly Progress Report.
2 Network Assumptions in this Dissertation

While one would desire the solution of a problem to be in its most generic form, in order to derive our theory some assumptions regarding the structure of the network had to be established. We first introduce two definitions and then present these assumptions.

Definition 1.1. Links or switches. In packet switching, since $\varphi > \pi$, the limiting resource is the propagation element. That is, forwarding nodes are assumed to have infinite resources and, therefore, they are not considered in the mathematical model. When developing the theory, we refer to a propagation element in the network as a link. When considering the distributed implementation of the protocol, switches are the network elements that connect links to each other.

Definition 1.2. Flow definition. A flow is defined as a set of packets that departure from the same source and arrive at the same destination after traversing the same set of links.

Assumption 1.1. Packet switching. We assume that the network operates as follows. Packets are generated by sources. Upon reception of a packet, a switch will forward it to the next link. Such recursive operation must guarantee (by some routing means) that packets are delivered at their correct destination. Destinations are responsible to receive packets.

Assumption 1.2. Longtime flows. This work assumes that flows have a long lifetime, when compared to the convergence properties of the network. This assumption is needed in order to guarantee that our network operates most of the time in an optimal steady state. Let $\tau_p$ be the time that it takes for a protocol to bring the network at its optimal
state. Let $\tau_j$ be the lifetime of a flow. Then $\tau_p = \tau_j$ means that the network operates most of the time in sub-optimal transient state, since every time a flow is terminated the protocol needs to bring the network to a new optimal state. Let us consider a flow to be longtime if $\tau_j > 10 \cdot \tau_p$. In actuality, routing protocols in IP networks update their routing tables approximately every 30 seconds. Therefore, as long as our protocol converges in less than 3 seconds this assumption must hold.

**Assumption 1.3. Cooperative users.** We assume that users cooperate in the sense that their actions are those specified by the protocol$^3$. Such assumption can be enforced by either dealing with users that well-behave or, otherwise, by deploying some type of policing nodes at the edge of the network that ensure their correct behavior.

**Assumption 1.4. Minimal switch intelligence.** We will assume that switches can have some degree of intelligence. While this design goes against the scalability principle - the more intelligence at the switch the more cost to process a packet and, therefore, the more difficult it is to grow the network -, we will guarantee that only very fast algorithms (with a logarithmic cost) are implemented at the switch node.

While connection oriented networks such as ATM [ATM96] or MPLS [MPL00] are the most suitable scenarios for these assumptions, the theory presented in this work can also be used in IP networks because, as mentioned above, actual IP routing table updates occur very infrequently (once every 30 seconds).

---

$^3$ An interesting game theoretic analysis arises when this assumption does not hold [AKE02].
3 Problem Statement

In this work, a network defines a set of bounded resources that are used by flows. Our objective is to identify the transmission rate for each flow that optimally utilizes the network resources. We define optimality in two senses:

- Network usage: rate allocations that are higher are preferable.
- Flow fairness: rate allocations that are fairer to each individual flow are preferable.

In this work, the lexicographic binary relation is the mathematical criterion that we choose in order to satisfy both usage and fairness conditions.

4 About this Dissertation

This dissertation is structured in seven chapters. The first chapter presents the introduction and motivates this work. The main body is included from chapter 2 to chapter 6. Chapter 2 presents a mathematical formulation of the problem. The problem is abstracted from any network arguments and only topological considerations are made. Under the assumptions of compactness and convexity, two results are presented. First, we prove that there must exist a solution and that such solution must be unique. Second, the equivalence lemma is introduced which shows that lexicographic and maxmin definitions are the same. Both results will be the starting point of the subsequent work. Chapter 3 presents our theory of unicast maxmin with minimal rate constraints. Two new optimality conditions are derived based on the concept of advertised rate: the bottleneck optimality condition and the projection optimality condition. A centralized algorithm that converges
to the maxmin solution is presented based on the optimality conditions. Then, we present a theory of maxmin bottleneck ordering. The precedent link relationship lemma proves the existence of a bottleneck order based on maxmin. This lemma allows us to define the CPG graph, which identifies such order, and to present an algorithm that constructs the graph. The theory of maxmin bottleneck ordering is useful in two ways: (1) to analyze the bottleneck structure of the network and (2) to measure the complexity of a maxmin algorithm. In chapter 4, we derive a practical protocol that implements maxmin with minimal rate constraints in a distributed manner. The algorithm has three modules that are respectively implemented at the source, switch and destination nodes. Convergence of the protocol for any arbitrary initial conditions is demonstrated and the protocol is empirically tested using a network simulator. Chapter 5 generalizes the previous maxmin theory to support the case of multicast communication. Likewise, optimality conditions, centralized algorithm, distributed protocol, convergence proofs and simulations are presented. In chapter 6, we study the maxmin problem for the case of discrete rates. While in previous chapters the feasible set was compact and convex, for the discrete case it is not. Therefore, the equivalence lemma does not hold. Moreover, the discrete lexicographic problem is proven NP complete. A new theory is provided to derive an optimality condition and develop a heuristic that converges to the lexicographically optimal or an almost-lexicographically optimal solution. In chapter 7, we briefly summarize the main contributions of this dissertation while comparing previous authors’ work. A section to describe future research in the area of concern is presented. Finally, an appendix can be found at the end that presents the implementation of the maxmin algorithms using Matlab language.
Chapter II

LEXICOGRAPHIC OPTIMIZATION: EXISTENCE, GLOBALITY AND UNIQUENESS

"Action sorts and grades; originally it knows only ordinal numbers, not cardinal numbers. But the external world to which acting man must adjust his conduct is a world of quantitative determinateness. [...] For acting man there exists primarily nothing but various degrees of relevance and urgency with regard to his own well-being."

*Human Action, Ludwig Von Mises (1949)*

1 Introduction

While in this work we develop a theory of network optimization, this chapter defines the lexicographic problem in a generic abstract form. In the first part, no data network arguments are used but only the topological structure of the problem is analyzed in order to study three properties: existence, globality and uniqueness. Only at the end of the chapter we apply the abstract lexicographic definition to the data network problem. Then we introduce the maxmin optimality condition and identify its relationship with the lexicographic definition.
Let us begin by introducing a set of definitions that will lead us to the concepts of preference and lexicographic optimization problems.

**Definition 2.1. Cartesian product, relation and binary relation.** The Cartesian product of \( n \) sets \( X_1, X_2, \ldots, X_n \), denoted as \( X_1 \otimes X_2 \otimes \ldots \otimes X_n \), is defined to be the set of all points \((x_1, x_2, \ldots, x_n)\) where \( x_i \in X_i \), \( 1 \leq i \leq n \). A relation is any subset of a Cartesian product. Similarly, a binary relation, denoted also as \( \leq_c \), is any subset of the Cartesian product of two sets \( X_1 \) and \( X_2 \). Given a binary relation set \( \leq_c \), we say that \( a \leq_c b \) if \((a, b)\) belongs to such set.

**Definition 2.2. Partial order.** Given a set \( S \), a binary relation \( \leq_c \) within the Cartesian product \( S \otimes S \) defines a partial order if it satisfies the following properties:

1. Reflexivity, \( a \leq_c a \) for all \( a \) in \( S \),
2. Antisymmetry, \( a \leq_c b \) and \( b \leq_c a \) implies \( a = a \),
3. Transitivity, \( a \leq_c b \) and \( b \leq_c c \) implies \( a \leq_c c \).

**Definition 2.3. Total order.** Given a set \( S \), a binary relation \( \leq_c \) within the Cartesian product \( S \otimes S \) defines a total order if it satisfies the following properties:

1. \( \leq_c \) satisfies the reflexivity, antisymmetry and transitivity properties,
2. **Comparability**, for any \(a, b \in S\), either \(a \leq_c b\) or \(b \leq_c a\).

As an example, let \(G\) be a directed graph. Let \(X\) and \(Y\) be two nodes in \(G\) and suppose that we define the relation \(g\) as:

\[
X \leq_g Y \iff \text{there exists a path from node } X \text{ to node } Y.
\]

Consider the graph examples in Figure 1. It is easy to see that the graph in Figure 1a defines a partial order. It does not define a total order since nodes \(C\) and \(D\) do not satisfy the comparability property. By adding two more edges, we obtain the graph in Figure 1b. It is easy to check that such graph defines a total order. Consider now the graph in Figure 1c. It does not define a partial order since nodes \(A\) and \(C\) do not satisfy the antisymmetry property (in general this is true for any directed graph with cycles). Finally, note also that \(g\) defines a binary relation in any of these graphs. For instance, the binary relation for graph in Figure 1a is: \{\((C,A), (C,B), (B,A), (D,B), (D,A)\)\}.

\[\text{Figure 1 Partial order and total order examples}\]

**Definition 2.4. Relationships.** Let \(a\) and \(b\) be two elements of \(S\). Then, we say that,

\[
a \geq_c b \text{ if } b \leq_c a
\]

\[
a =_c b \text{ if } a \leq_c b \text{ and } b \leq_c a
\]
\[ a \neq_c b \text{ if } a =_c b \text{ is false} \]

\[ a <_c b \text{ if } a \leq_c b \text{ and } a \neq_c b \]

\[ a >_c b \text{ if } b <_c a \]

**Definition 2.5. Convexity.** A set \( S \) is said to be **convex** if

\[
\begin{align*}
  x, x' \in S & \quad \lambda, \lambda' \geq 0 \\
  \Rightarrow & \quad \lambda \cdot x + \lambda' \cdot x' \in S \\
  \lambda + \lambda' & = 1 
\end{align*}
\]

where \( \lambda, \lambda' \in \mathbb{R} \).

**Definition 2.6.** The set \( S \) is compact if, from any sequence of elements \( x', x'', \ldots, x^{(i)} \) \ldots of \( S \), a subsequence can always be extracted that tends to some limit element \( x \) of \( S \).

**Definition 2.7.** Increasing permutation vector. Let \( x \) be a vector in \( \mathbb{R}^n \). We define its **increasing permutation vector** \( \bar{x} \) as a rearranged (permuted) version of \( x \) such that its components are in increasing order, i.e. \( \bar{x}(k) \leq \bar{x}(k+1) \) for all \( 1 \leq k < n \).

**Definition 2.8.** Increasing permutation set. Let \( S \) be a set of vectors in \( \mathbb{R}^n \). We define its **increasing permutation set** \( \bar{S} \) as the set of vectors such that for any \( x \in S \) there exists a vector \( y \in S \) such that \( x = \bar{y} \).

**Definition 2.9.** Lexicographic ordering. A vector \( y \) is said to be **lexicographically greater** or equal to \( y' \), denoted with \( y \geq_l y' \), if \( y_i < y'_i \) for some \( i \) implies \( y_k > y'_k \) for some
In addition, we say that \( y \) is **lexicographically** greater than \( y' \), denoted with \( y >_l y' \), if \( y \geq_l y' \) and \( y \neq y' \). Finally, we say that \( y \) is **lexicographically** equal to \( y' \) if \( y = y' \).

**Property 2.1. Total order lexicographic relation.** The lexicographic ordering defines a total order.

**Proof.** Without lost of generality, consider the particular case of the set of words in the Oxford English Dictionary. Notice that words in this dictionary are ordered following the lexicographic relation. Assume that the lexicographic relation is not a total order. Then there must exist an English word that cannot be found in order inside the dictionary. By contradiction, the property must be true.

**Definition 2.10. Lexmax optimization criterion.** Let \( S \) be a set of vectors in \( \mathbb{R}^n \). We say that \( x \) in \( S \) is a **lexmax** vector if \( x \) is lexicographically greater than \( x' \), for any other \( x' \) in \( S \) different from \( x \).

**Definition 2.11. Preference optimization problem (POP) and preference optimization value (POV).** The **preference optimization problem (POP)**, characterized by \( (\Pi, X, C) \), is defined as follows,

\[
POP : \quad \Pi x \rightarrow_c \text{ maximize} \\
\quad x \in X, \text{ where } X \text{ is convex and compact}
\]
where $\Pi x$ belongs to $\mathbb{R}^m$, $X \subset \mathbb{R}^n$, $\Pi$ is a mapping of $\mathbb{R}^n$ into $\mathbb{R}^m$, and $C$ is a binary relation. The set of vectors $x \in X$ is called the feasible set. Similarly, the set of points $y$ such that $y = \Pi x$ is called the transformed feasible set. The POP solution is the largest vector, in the $C$ sense, among all the points in the transformed feasible set. We refer to this vector as the *preference optimal value (POV).*

As an example, consider the following binary relation:

$$C \triangleq x >_C x' \iff f(x) > f(x')$$  \hspace{1cm} (2.3)

Then, Figure 2 presents three examples of POPs $(\Pi, X, C)$ in which $X$ is the set of real numbers $\mathbb{R}^1$, $\Pi$ is the identity transformation $\Pi x = x$, and $C$ depends on $f(x)$ as indicated by expression (2.3). In addition, $x$ is the POV for the POP in Figure 2b while $x'$ and $x''$ are the POVs for the POP in Figure 2c.

![Figure 2 POP examples](image)

Let us now introduce one type of POP, upon which our theory will be built.

**Definition 2.12.** *Lexicographic optimization problem (LOP) and lexicographic optimal value (LOV).* Let $\Gamma$ be a linear mapping and let $\overline{\Gamma}$ be the mapping obtained by reordering the output elements of $\Gamma$ in increasing permutation. The *lexicographic*
Optimization problem (LOP), characterized by \((\Gamma, X)\), is the POP obtained when \(C\) is the
lexicographic binary relation \((C = l)\), and \(\Pi = \Gamma\), for some \(\Gamma\). Similarly, the point
solution \(x\) of the LOP is called the lexicographic optimal value (LOV).

In the next three sections, we characterize the LOP problem in terms of three properties:
existence of a LOV, globality of a LOV and uniqueness of the LOV.

3 Existence in a LOP

Proposition 2.1. Any LOP problem \((\Gamma, X)\) has at least one LOV.

Proof. Note first that since \(\Gamma\) is a linear mapping and \(X\) is convex and compact, the set
produced by \(\Gamma x\) must also be convex and compact [BER63]. We note that a LOP
problem can be reduced to a set of maximization sub-problems. The first sub-problem
maximizes the value of the smallest element \(y_i\), with \(y \in \Gamma x\) and for any \(i\). More in
general, the \(k\)-th sub-problem maximizes the value of the \(k\)-th smallest element. Each one
of these sub-problems consists of finding the maximum in a compact subset of \(\mathbb{R}^1\);
therefore, for each one of them there exists a solution [RUD76]. It must follow that any
LOP must have a LOV.

§

4 Globality in a LOP

Property 2.2. The following is true,

\[
(1) \quad \bar{\lambda} \cdot x = \hat{\lambda} \cdot x
\]
where $x$ and $y$ are vectors in $\mathbb{R}^n$ and $\lambda$ is a nonnegative real number.

**Proof.** (1) is trivial if we consider that, for any arbitrary values of $a$, $b$ and $\lambda$ in $\mathbb{R}$, we have that $a \leq b \Rightarrow \lambda a \leq \lambda b$ if $\lambda \geq 0$. Let us now prove (2). We have that,

$$\left[ x + y \right]_1 = \bar{x} + \bar{y} = \min_{i=1,...,n} \{ x_i \} + \min_{i=1,...,n} \{ y_i \} = x_{i_1} + y_{i_2}$$

(2.4) for some $i_1$ and $i_2$ such that $1 \leq i_1, i_2 \leq n$. In addition, we have that,

$$\left[ x + y \right]_i = \min_{i=1,...,n} \{ x_i + y_i \} = x_{i_3} + y_{i_3}$$

(2.5) for some $i_3$ such that $1 \leq i_3 \leq n$.

Suppose that $\left[ x + y \right]_i > \left[ x + y \right]_j$, then $\min_{i=1,...,n} \{ x_i \} + \min_{i=1,...,n} \{ y_i \} > x_{i_1} + y_{i_2}$ which means that $x_{i_1} + y_{i_2} > x_{i_1} + y_{i_2}$ for any $j_1$ and $j_2$ such that $1 \leq j_1, j_2 \leq n$. But this last statement is a contradiction since it does not hold when $j_1 = j_2 = i_3$. Therefore, it must be that

$$\left[ x + y \right]_i \leq \left[ x + y \right]_j$$. If $\left[ x + y \right]_i < \left[ x + y \right]_j$, then $\bar{x} + \bar{y} \geq \bar{x} + \bar{y}$. Otherwise, we have that

$$\left[ x + y \right]_i = \left[ x + y \right]_j$$ and we can assume without loss of generality that $i_1 = i_2 = i_3$. In this case, let $x'$ and $y'$ be the vectors in $\mathbb{R}^{n-1}$ obtained by removing the $i_1$-th element from vectors $x$ and $y$, respectively, and let us iterate this proof substituting $x$ by $x'$ and $y$ by $y'$.

By induction, we conclude that (2) must also hold.

§
Definition 2.13. Improving direction. We say that \( u \in \mathbb{R}^n \) is an improving direction at point \( x \in X \) for a POP \((\Pi, X, C)\) if for any \( \lambda \in \mathbb{R} \), \( \lambda > 0 \), such that \( \Pi(x + \lambda \cdot u) \prec_c \Pi x \) there exists some \( \varepsilon \in \mathbb{R} \), \( 0 < \varepsilon < \lambda \), such that \( \Pi(x + \lambda \cdot u) \succ_c \Pi x \).

Definition 2.14. Unimodality. Let \((\Pi, X, C)\) be a POP and let \( x \) and \( y \) be two arbitrary points in \( X \) such that \( \Gamma y \succ_{\Gamma} \Gamma x \). Then, we say that \((\Pi, X, C)\) is unimodal if the straight-line direction from \( x \) to \( y \), that is \( u = y - x \), defines an improving direction at point \( x \) for POP \((\Gamma, X, C)\).

Let us return to our POP examples in Figure 2. By checking Definition 2.14, Figure 2a corresponds to a non-unimodal POP while Figure 2b and Figure 2c are unimodal POPs.

Lemma 2.1. Unimodality of LOPs. Any LOP is unimodal.

Proof. Let \((\Gamma, X)\) be a LOP and let \( x \) and \( x' \) be points in \( X \) so that \( \Gamma x \succ_{\Gamma} \Gamma x' \). Let also \( x'' \) be any point in-between the segment defined by \( x \) and \( x' \), that is \( x'' = (1 - \lambda)x + \lambda x' \) with \( 0 < \lambda < 1 \). In what follows, let \( y = \Gamma x \), \( y' = \Gamma x' \) and \( y'' = \Gamma x'' \).

Suppose that \( y \geq_i y'' \). Let \( r \) be an index such that \( y'_i = \bar{y}_i \) for \( i = 1, ..., r - 1 \) and \( y'_r > \bar{y}_r \). Note that such \( r \) must exist since \( \Gamma x' \succ_{\Gamma} \Gamma x \). Using Property 2.2 we have that,

\[
\bar{y}'' = \Gamma x'' = (1 - \lambda)\Gamma x + \lambda \Gamma x' \geq_i (1 - \lambda)\Gamma x + \lambda \Gamma x' = (1 - \lambda)\bar{y} + \lambda \bar{y}'
\]

(2.6)

If we define \( \bar{z} = (1 - \lambda)\bar{y} + \lambda \bar{y}' \) then (2.6) means \( \bar{y}'' \geq_i \bar{z} \) and hence,

\[
\bar{y} \geq_i \bar{z}.
\]

(2.7)
On the other hand,

\[ z_r = (1 - \lambda)\bar{y}_r + \lambda\bar{y}'_r > \bar{y}_r \]  

(2.8)

since \( \bar{y}'_r > \bar{y}_r \). Now let us consider expressions (2.7), (2.8) and the lexicographic definition. There must exist an index \( r' \) such that \( r' < r \) and \( \bar{y}_r > \bar{z}_{r'} \). This means that, for such index, \( \bar{y}_r > (1 - \lambda)\bar{y}_r + \lambda\bar{y}'_r \), but since \( \bar{y}'_r = \bar{y}_r \) we reach the contradiction \( \bar{y}_r > \bar{y}_r \).

From the above, we must conclude that \( \bar{y}'' > \bar{y} \) for any arbitrary \( y'' \), or equivalently, that any point between \( x \) and \( x' \) is greater in lexmax sense than \( x \). Therefore, the straight-line direction from point \( x \) to point \( x' \) must define an improving direction at point \( x \).

§

Proposition 2.2. Globality. Let \((\Pi, X, C)\) be a POP. If \((\Pi, X, C)\) is unimodal, then every local optimum in \((\Gamma, X, C)\) must be its global optimum.

Proof. The result builds on the fact that \( X \) is a convex set. For a formal proof, refer to theorem 1.1 in [PAP82].

§

Corollary 2.1. LOP Globality. Any local optimum of a LOP is its global optimum.

Proof. The proof is an immediate result of Proposition 2.2 and Lemma 2.1.

§
5 Uniqueness in a LOP

In the two previous sections we have proven that any POP problem has at least one solution (existence) and that the structure of the POP problem is of the unimodal type (globality). However, None of both conditions is sufficient to prove uniqueness. For instance, consider the POP in Figure 2c. It corresponds to a unimodal POP with multiple POVs (any $x$ such that $x' < x < x''$ is a POV). In this section, we present sufficient conditions for uniqueness.

Property 2.3. Uniqueness. Let $(\Gamma, X)$ be a LOP. If $\text{RANG}(\Gamma) = n$, where $n$ is as in Definition 2.11, then for any two LOVs $x$ and $y$, it must be that $y = x$.

Proof. Suppose that there exists $x$ and $y$ such that $x, y \in X$, $x \neq y$, and both are LOV.

Note first that $\Gamma x = \Gamma y$, otherwise one of them cannot be LOV. Let $i$ be such that $x_i \neq y_i$. Such $i$ must exists since $x \neq y$. Let $z$ be such that $z = \lambda \cdot x + (1 - \lambda) \cdot y$, for $0 < \lambda < 1$. Furthermore, let us choose $\lambda$ so that $[\Gamma z]_i \neq [\Gamma x]_j$, $\forall j$. Such $\lambda$ must exist since:

1. $\Gamma x \neq \Gamma y$, because $x$ and $y$ are linearly independent and $\text{RANG}(\Gamma) = n$ and

2. $[\Gamma z]_i = \lambda \cdot [\Gamma x]_i + (1 - \lambda) \cdot [\Gamma y]_i$ and $\lambda$ can take infinite number of values while $[\Gamma x]_i$ can only take at most $m$ different values.

Then,

$$\overline{\Gamma z} = \Gamma(\lambda \cdot x + (1 - \lambda) \cdot y) = \lambda \cdot \overline{\Gamma x} + (1 - \lambda) \cdot \overline{\Gamma y} \geq \lambda \cdot \overline{\Gamma x} + (1 - \lambda) \cdot \overline{\Gamma y}$$

(2.9)
Therefore,

\[ \Gamma z \succeq \Gamma x \]  \hspace{1cm} (2.10)

But \( \Gamma z \neq \Gamma x \) since \( [\Gamma z]_j \neq [\Gamma x]_j \) for all \( j \). Then, \( z \) is such that \( \Gamma z >_i \Gamma x \), which contradicts the original assumptions.

§

6 POPs, Ordinal Versus Cardinal Optimization

In calculus, one defines an optimization problem in terms of its objective function (also known in economics as the utility or satisfaction function). However, one can see that such approach introduces redundant parameters. Indeed, in order to define an optimization problem, it suffices to know what the order of preference among the elements in the feasible set is. Furthermore, in some instances the ordering preference, and not the objective function, may be the only information that one knows. Such observation has been made by few authors in the past (see [MIS49]). Debreu formally proved that it is possible to identify preference order relations for which there exists no numerical function representation [DEB54]. This means that one can always find optimization problems for which there exists no objective function representation and, therefore, traditional methods (e.g. differential calculus) cannot be utilized to solve them. Such is the case of the lexicographic order (see the proof provided in the footnotes of [DEB54]).
In this dissertation, we will provide a new theory to solve a set of optimization problems based on the lexicographic order. We apply this theory to the case of the network optimization problem, as defined in the following section.

7 Network Optimization and the Maxmin Problem

We define the network optimization problem as follows. We consider a set of \( n \) users and \( m \) resources that are physically distributed at different locations. Every time a user utilizes a unit of resource, he or she must lock it so that no other user can utilize it. In addition, resources are defined to be limited. The objective is to optimize the way users utilize the resources (Figure 3).

Two classic examples of network optimization problems are routing and flow control. In routing, a user’s objective is to identify those resources that connect him or her to another user within the minimum distance. In flow control, a user’s objective is to utilize those resources that connect him or her to another user allowing for the highest possible communication throughput.

![Figure 3 Network optimization problem](image-url)
One can parameterize a network optimization problem in terms of a POP. First, physical properties such as the location of users and resources, amount of resources or service requirements for each user (also known as quality of service or QoS parameters) define the feasible set $X$. Then, the binary relation $C$ is used to order the set of feasible solutions.

In this dissertation we consider a particular family of network optimization problems referred as \textit{maxmin} problems. Figure 4 presents an example. In the maxmin problem, users communicate between each other through a set of fixed paths. Each path includes a set of links or switches\footnote{we will equally use the term link or switch to denote a network resource.} in the network. A user $i$ has to decide at what rate $r_i$ he or she transmits the data, knowing that this rate consumes a certain number of resources in each of the traversed links. These conditions define the feasible set. Finally, in the maxmin problem, the binary relation $C$ is defined as the lexicographic relation, $C = l$.

\begin{figure}[h]
\centering
\includegraphics[width=0.8\textwidth]{maxmin_problem.png}
\caption{Maxmin problem}
\end{figure}
In the maxmin literature [BER92], the following is a well-known optimality condition.

*Definition 2.15. Maxmin optimality condition.* We say that a feasible point \( x \) in \( X \) satisfies the maxmin optimality condition if and only if for every feasible point \( x' \) in \( X \) such that \( x_i' > x_i \), for some \( i \), there exists an index \( k \) such that \( x_k \leq x_i \) and \( x_k > x_k' \).

The relationship between the maxmin optimality condition and the lexicographic definition has traditionally been overlooked. Because we will be frequently using this optimality condition, this chapter concludes showing how they relate to each other.

*Lemma 2.2. Equivalence.* Let \( x \) be an element in a compact and convex set \( X \). Then \( x \) is the largest element in \( X \) in the lexicographic sense if and only if \( x \) satisfies the maxmin optimality condition.

*Proof.* (only if part) Let \( x \) be the largest element in \( X \) in the lexicographic sense and suppose that \( x \) does not satisfy the maxmin optimality condition. Then, there must exist a rate vector \( x' \) in \( X \) with an index \( i \) such that \( x_i' > x_i \). Furthermore, there must exist no index \( k \) such that \( x_k \leq x_i \) and \( x_k > x_k' \). But this means that \( x' \) is larger in the lexicographic sense than \( x \), which is a contradiction.

(if part) Since \( X \) is convex and compact, there must exist one and only one element \( y \) in \( X \) that is the largest in the lexicographic sense among all the elements in \( X \). Assume that there exists an element \( x \) in \( X \) such that \( x \neq y \) and \( x \) satisfies the maxmin optimality condition. Now let \( u \) be the vector \( u = y - x \). Note that \( u \) must be an improving direction at point \( x \). Now consider \( z = x + \epsilon \cdot u \), with \( 0 \leq \epsilon \leq 1 \). Since \( x \) satisfies the maxmin
optimality condition, then if there exists an index \( i \) such that \( z_i > x_i \), there must exist another index \( k \) such that \( x_k \leq x_i \) and \( x_k > z_k \). But if this is true, then \( z \) cannot be lexicographically larger than \( x \), which is a contradiction. Therefore, it must be that \( x = y \).

In this dissertation we derive a lexicographic theory and we apply it to solve three different network topologies. From Chapter III to Chapter V we focus on unicast and multicast networks. As we will see, such networks define a compact and convex feasible set and, therefore, it is legitimate to use the maxmin optimality condition. In Chapter VI we focus on the case of optical networks, were bandwidth is measured in the discrete space of wavelengths. Such scenario defines a non-convex space and, therefore, the maxmin optimality condition cannot be used.
Chapter III

UNICAST MAXMIN THEORY

"Picture a pasture open to all. It is to be expected that each herdsman will try to keep as many cattle as possible on the commons. […]"What is the utility to me [the herdsman ] of adding one more animal to my herd?". The positive component is a function of the increment of one animal. Since the herdsman receives all the proceeds from the sale of the additional animal, the positive utility is nearly + 1. The negative component is a function of the additional overgrazing created by one more animal. Since, however, the effects of overgrazing are shared by all the herdsmen, the negative utility for any particular decision making herdsman is only a fraction of - 1. […]

The rational herdsman concludes that the only sensible course for him to pursue is to add another animal to his herd. And another…. But this is the conclusion reached by each and every rational herdsman sharing a commons. Therein is the tragedy. Each man is locked into a system that compels him to increase his herd without limit -- in a world that is limited"

The Tragedy of the Commons, Garret Hardin (1968)

1 Introduction

Because the available bandwidth in a network is limited, users have to restrict their network usage. Every link in a network has a capacity C. The sum of the rates in all the flows crossing the link must not exceed C. Now this leads to the problem of how to optimally distribute the limited bandwidth to the flows. On one hand, network utilization should be maximized, on the other hand, bandwidth assignment should provide fairness among all the users.

An intuitive definition of the maxmin solution is that it maximizes (max) the bandwidth assigned to those who are more poorly treated (min). That is, in maxmin we first
maximize the bandwidth assigned to that user who will get the less. Once this is done, we maximize the bandwidth assigned to the second user who will get the less. We continue to iterate in the same manner for all the users so that in the last iteration, the bandwidth assigned to the user who will get the most is maximized.

In this chapter we build our discussion on the maxmin problem for unicast networks. While this problem has been previously studied (e.g. [BER92]), our theory builds on two new results: the advertised rate of a link and the existence of a precedence link relationship. The former will lead us to two new maxmin optimality conditions while the latter will derive into a new theory of network bottlenecks.

2 The Constrained Maxmin Problem

In our theory, networks are made of two elements: links and flows. Links are responsible to transmit information from one location to another. A flow is a set of links that connect two or more distant users allowing the communication between them. Each flow transmits information at a certain rate and such action consumes a certain amount of resources in each one of the links in the flow. The amount of resources in each link is bounded.

Definition 3.1. The constrained maxmin problem (CMM) can be stated as follows,

$$\text{maximize } x(r) \text{ subject to } r \in X$$

(3.1)
where \( x(r) = \{ x_i(r) \} \) is a set of objectives defined inductively: \( x_i(r) = \min \{ r_i \mid i \in I \} \), and for \( k \geq 2 \), \( x_k(r) = \min \{ r_i \mid i \in I, r_i > x_{k-1} \} \) where \( I \) is the index set for the rate set \( r = \{ r_i \} \) and \( r_i \) is the rate for flow \( i \). The feasible set \( r \in X \) is described by:

\[
\begin{align*}
F_j \equiv \sum_{i \in V_j} r_i & \leq C_j \quad \forall j \in J \\
m_i \leq r_i \leq M_i & \quad \forall i \in I
\end{align*}
\] (3.2)

where \( J \) is the index set for the set of links, \( V_j \) is the set of flows crossing link \( j \), \( C_j \) is the given capacity for link \( j \), \( m_i \) is a given minimal flow rate constraint (MFR) for \( r_i \), and \( M_i \) is a given peak flow rate constraint (PFR) for \( r_i \). The maximization specified in equation (3.1) is of the lexicographic order: \( x_k \) must be maximized before \( x_{k+1} \), for all \( k \in L(r) \), where \( L(r) \) is the number of distinct rate values in the rate set \( r \). A rate set \( r \) is said to be feasible if it belongs to the constraint set \( X \).

Intuitively, the above formulation states that the smallest rate \( x_1(r) \) is to be maximized before the second smallest rate \( x_2(r) \) is maximized, and inductively, the maximization for a rate is carried out only after rates worse than the given rate have been maximized, hence the name maxmin. In the classic maxmin problem (MM), the bounds on the individual rate are relaxed to \( m_i = 0 \) and \( M_i = \infty \) [CHA95].

**Property 3.1. Lexicographic Optimality Condition.** A feasible rate vector \( r \) is an optimal solution to the CMM problem if and only if for every feasible rate vector \( y \), its increasing permutation \( \bar{r} \) (Definition 2.7) is lexicographically greater than or equal to \( \bar{y} \).
Proof. It is straightforward if we compare the definitions of lexicographic order and CMM problem.

Consider the sample network in Figure 1. \( r_1 = [65, 5, 45, 55, 75] \) and \( r_2 = [60, 10, 40, 60, 70] \) are two feasible rate vectors. Their corresponding increasing permutations are \( \bar{r}_1 = [5, 45, 55, 65, 75] \) and \( \bar{r}_2 = [10, 40, 60, 60, 70] \). Now because \( \bar{r}_2 >, \bar{r}_1 \) we conclude that \( \bar{r}_1 \) is not the CMM solution.

![Figure 1 Network example with minimal and maximal rate constraints](image)

3 Extended Network

Definition 3.2. Extended network. Let \( N \) be a network. We define its extended network as the network obtained by executing the next two steps:

1. Consider flow \( i \) in \( N \). If \( M_i < \infty \) then add a new link \( j \) with capacity \( C_j = M_i \) at one of the termination points of flow \( i \). Repeat this process for all \( i \).

2. Set all the peak flow rate constraints to infinity.

Figure 3 shows the transformation that must be applied to any of the flows in the original network.
Property 3.2. Extended Equivalent Problem. The solution of the CMM problem for $N$ is the same as that of $N^*$. 

Proof. It is straightforward if we note that the feasible set of the CMM problem is preserved by the transformation.

Note also that the extended network defines an idempotent operation, since the transformation only applies to those flows whose PFR are not infinity. In other words, the extended network of a network with infinite PFR values is the network itself.

From this point on, we will assume without loss of generality that the input to our maxmin problem is an extended version of the actual given network. That is, all the PFRs are assumed infinity.

4 Maxmin Theory

The following is a well-known maxmin optimality condition [BER92].

Property 3.3. Maxmin Optimality Condition. A feasible rate vector $r$ is an optimal solution to the CMM problem if and only if for every feasible rate vector $r'$ with $r_i' > r_i$, for some $i$, then there exists a flow $k$ such that $r_k \leq r_i$ and $r_k > r_k'$. In other words, a rate
vector $r$ is maxmin if it is feasible and for each flow $i$, $r_i$ cannot be increased while maintaining feasibility without decreasing $r_k$ for some flow $k$ for which $r_k \leq r_i$.

**Proof:** By using Lemma 2.2, this property must be true if we notice that the CMM problem can be written in terms of a LOP $(\Gamma, X)$ where $X$ is compact and convex and $\Gamma$ is the identity mapping $\Gamma x = x$.

Both the maxmin and the lexicographic optimality conditions (Property 3.1 and Property 3.3) are not automatable as they require excessive (exponential) amount of checking, making them unsuitable for computer implementation. The theory presented in this section introduces two new automatable optimality conditions that build on the concept of *advertised rate*.

**Definition 3.3. Advertised rate.** Let $Vm_j$ denote the set of flows crossing link $j$ with the condition $r_i = m_i$. The quantity $R_j$ defined below is called the *advertised rate* for link $j$:

$$R_j = \begin{cases} \max \{r_i \mid i \in V_j, r_i > m_i\} & \text{if } Vm_j \neq V_j, F_j = C_j \\ \infty & \text{if } F_j < C_j \\ 0 & \text{if } Vm_j = V_j, F_j = C_j \end{cases}$$

(3.4)

**Definition 3.4. Bottleneck Link Constraint.** Flow $i$ crossing link $j$ is said to be *bottleneck link constrained* (BLC) at this link $j$ if $F_j = C_j$, $r_i > m_i$, and $r_i = R_j$. 

34
Definition 3.5. Minimum Rate Constraint. Flow $i$ crossing link $j$ is said to be Minimum Rate Constrained (MRC) at this link $j$ if $F_j = C_j$, $r_i = m_i$ and $r_i \geq R_j$.

Definition 3.6. Constrained and Unconstrained Flows. Flow $i$ crossing link $j$ is said to be a constrained flow at link $j$ if

1. Flow $i$ is BLC at link $k \neq j$; or

2. Flow $i$ is MRC at some link.

Similarly, flow $i$ at link $j$ is said to be an unconstrained flow at link $j$ if it is not constrained at the same link. This definition arises from the fact that flow $i$ is unconstrained from link $j$ point of view because flow $i$ is not constrained by some other means to set its rate lower (BLC) or higher (MRC) than its own advertised rate.

The main difference between the above bottleneck definitions and those of other authors (e.g. [HOU98]) is that we impose more technical constraints such as the advertised rate conditions to ensure that the bottleneck link constraint and the minimal rate constraint are treated separately. This turns out to be the key in the design of simpler optimality conditions.

Definition 3.7. Saturation. A link $j$ is said to be saturated if the total flow is equal to its capacity: $F_j = C_j$.

Definition 3.8. Pseudo-saturation. A link $j$ is said to be pseudo-saturated if every flow crossing this link is constrained at it and its capacity is not fully utilized: $F_j < C_j$. 
In the definition of advertised rate (Definition 3.3), the first case is the ordinary case where link $j$ has at least one unconstrained flow. The second case is one in which link $j$ is pseudo-saturated. The third case is a pathological case where all flows are at their respective minimum rates and the link is saturated. In this case, the entire available bandwidth is used by all the flows that are not allowed to drop their rates. Notice that we do not define $R_j$ for the case where $F_j > C_j$, since this is a case where no feasible solution exists.

Intuitively, the advertised rate of a link is the largest rate of the flows crossing it with rates higher than their respective MFRs so that the link is saturated. If no flow can be allocated a higher rate than its MFR rate, the advertised rate is defined to be zero. If saturation is not possible, the advertised rate is defined to be infinity.

**Definition 3.9. Fair Share.** The ratio $C_j/N_j$ is called the fair share at link $j$. Intuitively, it is the fair share for all the flows crossing link $j$ while assuming none of them is constrained.

**Definition 3.10. Remaining Fair Share.** The ratio $(C_j - Fc_j)/(N_j - Nc_j)$ is called the remaining fair share at link $j$, where $Fc_j$ is the aggregated rate of the constrained flows at link $j$ and $Nc_j$ is the number of constrained flows at link $j$, if $N_j - Nc_j > 0$. Intuitively, the remaining fair share at a link is the fair share after taking the bandwidth of its constrained flows.

Let us now use the above definitions to state two new optimality conditions.
Theorem 3.1. Bottleneck Optimality Condition. A rate vector $r$ is a solution to the CMM problem if and only if for every flow, it is either BLC or MRC at some link.

Proof. (Only-If part): Suppose that the rate vector $r$ is maxmin and, to arrive at a contradiction, assume that there exists a flow $i$ that is not BLC nor MRC at every link of the flow. Consider then any link $j$ in the path of flow $i$. There are two cases: (1) Link $j$ is saturated or (2) Link $j$ is not saturated. Consider case (1) where link $j$ is saturated. Then either (1.1) $r_i = m_i$ or (1.2) $r_i > m_i$. Assume case (1.1), then, since flow $i$ is not MRC, we have $r_i < R_j$. Now assume two more cases: (1.1.1) $Vm_j \neq V_j$ and (1.1.2) $Vm_j = V_j$. If $Vm_j \neq V_j$, then because of saturation assumption, there must exist a flow $k$ crossing link $j$ such that $r_k = R_j$ and $r_k > m_k$. Since $r_i < R_j$ then we have that $r_k > r_i$. Thus, we can increase $r_i$ without violating the capacity constraint by dropping $r_k$. Assume $Vm_j = V_j$ (1.1.2), but then $R_j = 0$ and $r_i < R_j = 0$ is not possible. Assume now case (1.2). Then $r_i < R_j$, since $r_i \neq R_j$ because flow $i$ is not BLC and $r_i \leq R_j$ by definition of $R_j$. Then, there must also exist a flow $k$ crossing link $j$ such that $r_k = R_j$ and $r_k > m_k$. Since $r_i < R_j$ then we have that $r_k > r_i$. Thus, we can increase $r_i$ without violating the capacity constraint by dropping $r_k$. Finally, consider case (2) where link $j$ is not saturated. Then we can increase $r_i$ without dropping any other flow's rate to reach the capacity.

Now from the above, we can increase $r_i$ at every link that flow $i$ crosses by a non-zero positive amount without having to decrease the rate of an $r_k$ such that $r_k \leq r_i$ while maintaining feasibility. This contradicts the optimality condition of the rate vector $r$. 

37
(If part) Conversely, suppose that rate vector \( r \) satisfies the bottleneck optimality condition. Now, for every flow \( i \), there are two cases to consider: (1) it is MRC at link \( j \), or (2) it is BLC at link \( j \). Assume case (1), now there are two sub-cases to consider: (1.1) link \( j \) is saturated and (1.2) link \( j \) is not saturated. Consider case (1.1), if we increase the rate \( r_i \), we must drop some other rate in order to satisfy the capacity constraint. We cannot drop any flow whose rate is at its minimum. Thus the only rate we can drop must be from any flow \( k \) with the constraint \( r_k > m_k \) and, hence, \( r_k \leq R_j \). But by the definition of MRC, we have \( r_j \geq R_j \). Thus, if we increase \( r_i \), we must decrease some \( r_k \) with the condition that \( r_j \geq r_k \). Consider case (1.2), by the definition of the advertised rate and MRC, we must have \( r_i = m_i \geq \infty \), which is unfeasible. Finally, assume case (2), flow \( i \) is BLC at a link \( j \). Thus, in order to increase \( r_i \) without violating the capacity constraint, one must decrease a rate \( r_k \) such that \( r_k > m_k \) and \( r_k \leq r_i \), where flow \( k \) crosses link \( j \).

\[ \text{Theorem 3.2. Projection Optimality Condition.} \]  
A rate vector \( r \) is a solution to the CMM problem if and only if for every flow \( i \) its rate satisfies the following condition,

\[ r_i = \max \{ \min \{ R_j : j \in P_i \}, m_i \}, \quad (3.5) \]

where \( P_i \) is the set of links in the path of flow \( i \). We call the above expression the projection optimality condition.

\[ \text{Proof.} \] (only if part) Assume \( r \) is a solution to the CMM problem and let \( i \) be an arbitrary flow. Then there are two cases to consider: (1) flow \( i \) is MRC at link \( k \), (2) flow
\(i\) is BLC at link \(k\). Assume case (1). Then because \(r_i = m_i\) and \(r_i \geq R_k\) we have that (3.5) must be true. Consider case (2). There are two cases: (2.1) \(R_k = \min\{R_j : j \in P_i\}\) or (2.2) \(R_k \neq \min\{R_j : j \in P_i\}\). Assume case (2.1), then (3.5) is true since feasibility condition implies \(r_i \geq m_i\) and BLC means \(r_i = R_k\). Assume case (2.2), then exists a link \(k'\) such that \(R_{k'} < R_k\) and flow \(i\) crosses this link. There are two more cases: (2.2.1) \(r_i = m_i\) and (2.2.2) \(r_i > m_i\). If \(r_i = m_i\) (2.2.1), since \(r_i = R_k > R_{k'}\) then flow \(i\) is MRC at link \(k'\) and the case is reduced to that of case (1). If \(r_i > m_i\) (2.2.2), then \(R_{k'} \geq r_i = R_k\), which is a contradiction.

(If part) Consider the two cases (1) \(R_k = \min\{R_j : j \in P_i\} > m_i\) and (2) \(R_k = \min\{R_j : j \in P_i\} \leq m_i\). Suppose case (1), then from (3.5) \(r_i = R_k > m_i\) and \(F_k = C_k\) (otherwise \(R_k = \infty\)), which means that flow \(i\) is BLC at link \(k\). Suppose now case (2). Then from (3.5) we have \(r_i = m_i \geq R_k\) and \(F_k = C_k\) (otherwise \(R_k = \infty\)), which means that flow \(i\) is MRC at link \(k\).

5 CPG Algorithm

We now show how the theory exposed in the previous sections can be used to design a centralized algorithm that solves the CMM problem. Let us for now claim that the algorithm presented in section 5.2 can solve any CMM problem by reducing it to solving the single-link network case. This justifies our necessity to design an algorithm that solves this case.
5.1 Single-link Algorithm

One can use Theorem 3.2 as a method to check whether a solution is optimal or not. That is, given a set of rates $r$ we first calculate the advertised rate as expression (3.4) indicates and then we check if any rate $r_i$ satisfies the projection optimality condition, $r_i = \max\{\min\{R_j : j \in P_i\}, m_i\}$. If it does, then $r$ is optimal, otherwise it is not. This suggests an algorithm based on an update-and-check scheme. That is, we keep updating the rates until the projection optimality condition is satisfied. Intuitively, in order to ensure convergence, the update of a rate has to be done so that the rate vector at each iteration is “closer” to the maxmin solution than the rate vector at the previous iteration. Figure 3 presents the pseudocode of this algorithm.

\textbf{SolveSingleLinkCMM} (N)

\textbf{Input parameters:}

$N$: a network with a single link $j$

\textbf{Output parameters:}

$\{r_i\}$: set of maxmin rates

$R_j$: advertised rate for link $j$

1. set $l = 1$; Mark all flows as unconstrained;

2. For any unconstrained flow $i$ set $r_i' = (C_j - Fc_j')/(N_j - Nc_j')$;

3. Calculate $R_j'$. If $r_i' = \max\{R_j', m_i\}$ for each flow $i$ then set $R_j = R_j'$, set $r_i = r_i'$ for all $i$ and stop; otherwise, set $r_i' = \max\{R_j', m_i\}$ for any flow $i$ such that $r_i' \neq \max\{R_j', m_i\}$, mark these flows as constrained, do $l = l + 1$ and go to step 2;

\textbf{Figure 3 Single link projection algorithm (see Matlab function RATEsinglelink() in Appendix)}

In the above procedure, we use a super-index for the values of $Fc_j$, $R_j$ and $r_i$ to indicate the iteration at which they are computed. Intuitively, at step (2) the rate
assignment advances towards the maxmin solution. Within this step, rates may move out of the feasible set, in which case at step (3) they are projected back to it. The process is iterated until the projection optimality condition is satisfied.

Before continuing with the convergence proof, let us state a well-known algebraic property.

Property 3.4. Algebraic equalities and inequalities. The following is true,

1. \( \frac{x + |x|}{y + |y|} > \frac{x}{y} \) if and only if \( \frac{x}{y} < \frac{|x|}{|y|} \)

2. \( \frac{x + |x|}{y + |y|} = \frac{x}{y} \) if and only if \( \frac{x}{y} = \frac{|x|}{|y|} \)

3. \( \frac{x - |x|}{y - |y|} > \frac{x}{y} \) if and only if \( \frac{x}{y} > \frac{|x|}{|y|} \)

4. \( \frac{x - |x|}{y - |y|} = \frac{x}{y} \) if and only if \( \frac{x}{y} = \frac{|x|}{|y|} \)

Proof. The proof of the above property is not relevant to the purpose of this theory and we leave it to the reader.

§

Proposition 3.1. Convergence of the Single-link Projection Algorithm. The single-link projection algorithm converges to the CMM solution in a finite number of steps.
Proof. Note that if the algorithm finishes, then the solution is optimal since it satisfies the projection optimality condition. Therefore, it suffices to show that the algorithm terminates. It is also worth noticing that because $N$ is a single-link network, the set of constrained flows can only include MRC flows. In addition, note that since at step (2) the non-constrained flows are assigned the remaining fare share, then $F_j = C_j$ is true and, therefore, $R_j$ computed in step (3) is bounded, $R_j < \infty$. Now to prove convergence, consider the first iteration at step (2). Since initially all flows are unconstrained, they will be assigned the same fair share rate. Now at step (3) there are two cases: (1) $m_i \leq r_i$ for all flows and (2) $m_i > r_i$ for some flow $i$. Assuming case (1), then from the advertised rate definition we have that $r_i = R_j$ for any flow $i$ and the algorithm stops. Consider now case (2). Since the rates for those flows such that $m_i > r_i$ are projected to $m_i$ (step 3), which has a larger value than that of the current fair share, at the next iteration the remaining fair share (step 2) will take a smaller value (here we use statement 3 in Property 3.4). Therefore, those flows that become minimal rate constrained will continue to be in that condition at the next iteration. This shows that the number of MRC flows does not decrease while iterating the algorithm. If at some iteration such number remains the same, then the algorithm terminates. Finally, since the number of flows is bounded, the algorithm must terminate.

§

As an example, assume the single-link network configuration in Figure 4. It consists of three flows, two of them are minimal rate constrained and one is not.
Figure 4 Single-link network example

Figure 5 presents the execution of the \textit{SolveSingleLinkCMM} algorithm for this particular network. The plane shows the feasible space (the set of solutions that are feasible). The line shows the evolution of the rate vector as it converges to the CMM solution. Initially, it is set to zero. Then at each iteration the rate vector steps into two new points, corresponding to steps (2) and (3) of the algorithm. Note how the projection algorithm converges to the optimal solution by projecting the rate vector to the feasible space whenever it moves out of it. The solution obtained corresponds to the maxmin rate allocation \( r_1 = 100, r_2 = 40, r_3 = 10 \).

Property 3.5. Monotonic rate behavior in the single-link algorithm. Consider the single-link algorithm and let \( l \) be such that \( l + 1 \leq L \), where \( L \) is the number of iterations that it takes to terminate the procedure. Then the following must be true,

1. \( F_{c_j}^l < F_{c_j}^{l+1} \),

43
2. \( R_j^i > R_j^{i+1} \),

3. \( r_i^i > r_i^{i+1} \) if flow \( i \) is unconstrained at iteration \( i+1 \),

Now let \( \{ r_i \} \) be the maxmin rate solution of a single-link network \( N \) and let \( R_j \) be its corresponding advertised rate. Consider the network \( N' \) obtained by removing flow \( k \) from \( N \) and subtracting a value \( x \) from the link capacity. That is, the link capacity of the new network, \( C' \), takes the value of \( C' = C - x \). \( x \) is chosen so that \( N' \) defines a non-empty feasible set. Let \( R'_j \) be the advertised rate that corresponds to the maxmin solution of the new network. The following must be true,

4. if \( x < r_k \), then \( R'_j > R_j \)

5. if \( x = r_k \), then \( R'_j = R_j \)

6. if \( x > r_k \), then \( R'_j < R_j \)

Proof. Statement 1 is true since at each iteration the number of constrained flows increases. Consider now statement 2. At step 3 of the single-link algorithm we have that \( F_j = C_j \) and, therefore, the advertised rate at this step must be \( R_j^i = (C_j - Fc_j^i) / (N_j - Nc_j^i) \) (from Definition 3.3). By definition of constrained flow at its minimal rate, it must also be true that \( (Fc_j^{i+1} - Fc_j^i) / (Nc_j^{i+1} - Nc_j^i) > R_j^i \). Then using statement 3 in Property 3.4 it must be that \( R_j^i > R_j^{i+1} \). Statement 3 follows from statement 2, since by construction of the algorithm we have that \( r_i^i = R_j^i \).
Let us now consider statement 5. Consider the network $N'$ and let $\{r'_i, i \neq k\}$ be the rates assigned to its flows. Note that if $\{r'_i = r_i, i \neq k\}$ then the link is saturated, $F_j = C_j$. Furthermore, making $R'_j = R_j$ and checking the projection optimality condition, the rates $\{r'_i = r_i, i \neq k\}$ are maxmin. Because the solution is unique, statement 5 must hold.

Assume now statement 4. Consider $M$ to be the network obtained from statement 5. Then, one can obtain network $N'$ in statement 4 by adding a capacity of $r_k - x$ to network $M$. As proved in statement 5, the advertised rate of network $M$ is $R_j$. Let $S$ be the set of flows that in network $M$ are constrained but that in network $N'$ are not. Note that since $N'$ is the same network as $M$ but with some additional capacity, $S$ may not be empty. Now we have that,

$$R'_j = \frac{C'_j - Fc'_j}{N'_j - Nc'_j} = \frac{C_j + (r_k - x) - (Fc_j - \sum_{i \in S} m_i)}{N_j - (Nc_j - |S|)} = \frac{C_j - Fc_j + (r_k - x) + \sum_{i \in S} m_i}{N_j - Nc_j + |S|},$$

since

$$\frac{(r_k - x) + \sum_{i \in S} m_i}{|S|} > R_j = \frac{C_j - Fc_j}{N_j - Nc_j},$$

then from statement 1 in Property 3.4 it must be that $R'_j > R_j$.

Finally, let us consider statement 6. One can represent $R'_j$ as a function of $x$, i.e. $R_j = f(x)$. Now note that the higher $x$ is, the smaller the capacity of the link is and,
hence, the smaller the advertised rate is. Therefore, \( R_j' \) is a monotonic function of \( x \) that increases as \( x \) decreases. Since statement 5 is true, statement 6 must follow.

§

5.2 Multi-link Algorithm

In this section, we present the Constrained Precedence Graph (CPG) algorithm that we will use to solve the general multi-link problem (Figure 6). The algorithm introduces two new variants when compared to the maxmin algorithms previously published (e.g. [HOU98-ARU96]). First, the CPG procedure can be executed in a parallel machine architecture because step 3 can be carried out independently at each link. Second, the algorithm does not depend on the feasibility constraints nor the fairness objectives, both dependencies are abstracted into the SolveSingleLinkCMM procedure. While this was previously defined according to the maxmin criterion and the MFR constraints (Figure 3), one could redesign it to accommodate a different optimization definition with different constraints and still expect the CPG algorithm to find the solution for the multi-link network case.

\[
\text{SolveNetworkCMM (N)}
\]

\textit{Input parameters:}
\begin{itemize}
  \item \( N \): a network
\end{itemize}

\textit{Output parameters:}
\begin{itemize}
  \item \( \{r_i\} \): set of maxmin rates
  \item \( \{R_j\} \): set of advertised rates
\end{itemize}

1. \( L = 1 \);
2. For each single link \( N_j \) in \( N \), obtain \( R_j \) and \( \{r_i \mid i \in V_j\} \) by executing SolveSingleLinkCMM( \( N_j \));
3. For each link \( j \) in \( N \) such that \( R_j = \min \{R_k \mid \text{link } j \& \text{ link } k \text{ share a joint flow}\} \), do the following:
   3.1 Add link \( j \) to the set of level \( L \) links;
   3.2 Update \( C_i \) for all link \( k \) sharing a joint flow with link \( j \) doing the following \( C_i = C_i - \sum_{v_j \cap v_k} r_i \);
3.3 Remove link j and all the flows crossing link j from N;
4. If N is not empty, do $L = L + 1$ and go to step 1; otherwise stop;

Figure 6 SolveNetworkCMM algorithm (see Matlab function CPG() in Appendix)

Before presenting the convergence proof of the CPG algorithm, the following property shows the monotonic behavior of the advertised rate.

**Property 3.6. Monotonic rate behavior in the multi-link algorithm.** Let $R_i^x$ be the computed advertised rate at step 2 in the CPG algorithm for link $x$ at iteration $y$. Then,

$$R_j^l \leq R_j^{l+1} \quad (3.6)$$

*Proof.* Let $C_j^l$ be the available bandwidth at iteration $l$ for a link $j$ that is not removed at this iteration. There exist two cases: (1) no flow crossing link $j$ is removed from the network at iteration $l$ or (2) some flow crossing link $j$ is removed from the network at iteration $l$. If (1) is true, then there is no change at iteration $l$ from link $j$ standpoint and $R_j^l = R_j^{l+1}$. If (2) is true, then let flow $i$ be such that it crosses link $j$, is removed from the network at iteration $l$, and it is bottlenecked at link $k$. Let $r'_i(u)$ be the rate assigned to flow $i$ by the SolveSingleLinkCMM procedure at step 2 of the CPG algorithm at level $l$ and executed at link $u$, with flow $i$ crossing link $u$. It must be that $r'_i(k) \leq r'_i(j)$, otherwise flow $i$ would not be bottlenecked at link $k$. Now from statements 4 and 5 in Property 3.5, it must be that $R_j^l \leq R_j^{l+1}$. §
Property 3.7. CPG Convergence. The rates computed by the CPG algorithm converge to the maxmin solution in a finite number of steps.

Proof. The property must hold if the following two conditions are true:

1. Correctness. If the CPG finishes, then the output rates are optimal.

2. Finiteness. The CPG finishes in a finite number of steps.

The execution of the procedure SolveSingleLinkCMM together with the minimization statement at step 3 guarantee that, upon termination of the CPG algorithm, every flow is either BLC or MRC. Therefore, the proof of correctness follows from the bottleneck optimality condition stated in Theorem 3.1.

Consider now finiteness. Note that if the single-link subroutine can be properly executed then at every loop of the CPG algorithm at least one link is removed. If this is the case, the algorithm finishes with at most \( n \) iterations, where \( n \) is the number of links in the network. Hence, we only need to show that the single-link subroutine can be executed properly at each iteration. Now examining the operations from its pseudocode in Figure 3, it can be seen that the algorithm executes correctly if and only if the minimal rate constraints are feasible when compared to the link capacity. That is,

\[
\sum_{i \in V_j} m_i \leq C^l_j, \text{ at any arbitrary level } l
\]  

(3.7)

Therefore, we need to show that when the CPG algorithm fixes some rates at a level \( l \), the remaining links in the network satisfy equation (3.7) so that the single-link subroutine can be properly executed at level \( l+1 \). To see this, consider any arbitrary link \( j \) that is
removed from the network at level $l$ and let $k$ be an arbitrary link that is not removed at any of the levels $l$, ..., $L$. If $V_j \cap V_k = \emptyset$ then condition (3.7) is satisfied, since there is no change in the single link problem for link $k$ at level $l$. If $V_j \cap V_k \neq \emptyset$ then $R_j^l \leq R_k^l$ must hold. Let $r_i^l(u)$ be defined as in the previous proof and let $i$ be an arbitrary flow that is removed at iteration $l$ and that crosses both links $j$ and $k$. Then there are two cases: (1) $r_i^l(j) = m_i$ and (2) $r_i^l(j) = R_j^l$. Consider case (1). Since it is at its minimum rate, it must be that $r_i^l(j) \leq r_i^l(k)$. Assume now case (2). We have $r_i^l(j) = R_j^l \geq m_i$ which means $R_k^l \geq m_i$. Then it must be that $r_i^l(k) = R_k^l$ which means that $r_i^l(j) \leq r_i^l(k)$. Therefore, we have proved that in all the cases $r_i^l(j) \leq r_i^l(k)$. This means that feasibility is ensured at level $l+1$ at link $k$ since the new settled rate, $r_i^l(j)$, is not bigger than a rate that is already feasible, $r_i^l(k)$, and since we chose $i, j$, and $k$ to be arbitrary indexes.

Consider as an example the sample network in Figure 1. Figure 7 presents its extended version.

\[
\begin{array}{c|c|c|c}
\text{Flow} & \text{MRC} & \text{PRC} \\
\hline
1 & 60 & \infty \\
2 & 0 & \infty \\
3 & 0 & \infty \\
4 & 55 & \infty \\
5 & 0 & \infty \\
\end{array}
\]

\textit{Figure 7 Extended network for network in Figure 1}
Now executing the CPG algorithm we obtain the set of solutions,

\[ r_1 = 60, r_2 = 10, r_3 = 40, r_4 = 60, r_5 = 70 \text{ Mbps} \]  (3.8)

And the advertised rates for the maxmin solution are,

\[ R_1 = 10, R_2 = 40, R_3 = 60, R_4 = \infty, R_5 = 70 \text{ Mbps} \]  (3.9)

Let us now check the optimality of this solution. Consider the bottleneck optimality condition. In this case, note that flow 1 is MRC at link 1, flow 2 is BLC at link 1, flow 3 is BLC at link 2, flow 4 is BLC at link 3 and flow 5 is BLC at link 5. Since every link is BLC or MRC at some link, we can conclude that (3.8) is the maxmin solution. Note also that no flow is BLC nor MRC at link 4, which shows that such link is pseudosaturated.

One can also prove the optimality of our solution by using the projection optimality condition. Likewise, it can be easily checked (we leave it to the reader) that for every flow, its allocated rate is the maximum of its own minimal rate and the minimum of the advertised rates among the links it crosses.

6 Theory of Maxmin Bottleneck Ordering

When solving the maxmin problem, the converge speed of each advertised rate differs among the links in the network. Convergence time wise, one may think that a link \( i \) precedes link \( j \) if link \( i \) requires “less time” to converge than link \( j \). The pitfall of this definition is that it depends on time properties of the network such as propagation, queuing or processing delays, rather than on its actual topological bottleneck characteristics.
In this section we shall derive a theory of bottleneck ordering based on the topological bottleneck characteristics of the network.

6.1 Bottleneck Maxmin Order Definition

We first show the existence of a quasi-order relation within the set of links. Then we define the concept of order equality. Both will lead us to the maxmin partial order definition. In what follows, let $L$ be the set of links in a network.

**Definition 3.11. maxmin-faster.** We say that link $i$ is maxmin-faster than link $j$ when link $j$ cannot converge if link $i$ has not converged. We denote this relationship using $i <_m j$.

**Property 3.8. Maxmin quasi-order.** The maxmin-faster definition yields a quasi-order relation since it satisfies the anti-reflexive and transitive properties [WRI92],

1. **Anti-reflexive:** $i <_m i$ is false for all $i$ in $L$.

2. **Transitive:** $i <_m j$ and $j <_m k$ imply $i <_m k$.

Let us now introduce the equally maxmin-faster definition.

**Definition 3.12. Equally maxmin-faster.** We say that link $i$ is equally maxmin-faster than link $j$ if the set of links that have to converge before link $i$ can converge is equal to the set of links that have to converge before link $j$ can converge. More formally,

$$i =_m j \text{ if and only if } \{ m \in L \mid m <_m i \} = \{ n \in L \mid n <_m j \}$$  \hfill (3.10)

We can now define a partial ordering in the set of links.
Definition 3.13. Maxmin faster or equally faster. We say that link \(i\) is maxmin-faster or equally faster than \(j\) if \(i \leq_m j\) or \(i =_m j\). We denote this relation using \(i \leq_m j\).

Property 3.9. Maxmin partial order. The maxmin-faster relation defines a partial order since it satisfies the reflexive, anti-symmetric and transitive properties [WRI92],

1. Reflexive: \(i \leq_m i\) for every \(i\) in \(L\).

2. Anti-symmetric: \(i \leq_m j\) and \(j \leq_m i\) imply \(i =_m j\).

3. Transitive: \(i \leq_m j\) and \(j \leq_m k\) imply \(i \leq_m k\).

The previous definitions show that it is possible to find a partial order within the set of links in a network. In what remains, we provide a methodology to find such order.

6.2 Precedent Link Relationship

The topological structure of the network together with the available number of resources (bandwidth) at each link define the fundamental relationships between the links in terms of their convergence time. As it will be seen in this section, they (topology and amount of resources) uniquely define the bottleneck ordering.

Lemma 3.1. Precedent link relationship. Let \(N\) be a network and let \(j \in N\) be a link that is removed at iteration \(l+1\) of the centralized maxmin algorithm. Let also \(i\) be a link in \(N\) that at iteration \(l\) shares a flow with link \(j\) so that \(R_i^l < R_j^l\). Note that such a link must exist, otherwise link \(j\) would be removed at iteration \(l\). Then, it must be that either link \(i\)
becomes a bottleneck at iteration $l$ or there exists at least one link $k$ that shares at least one flow with link $i$ such that it becomes a bottleneck at iteration $l$.

**Proof.** We will prove the above proposition by contradiction. Suppose that under the stated conditions, neither link $i$ becomes a bottleneck at iteration $l$ nor there exists a link $k$ that shares at least one flow with link $i$ such that it becomes a bottleneck at iteration $l$. Then, since it does not become a bottleneck, link $i$ is not removed from the network at iteration $l$. Furthermore, since no other link that shares a flow with link $i$ becomes bottlenecked at iteration $l$, it must be that $R_i^{l+1} = R_i^l$. Since also $R_j^l \leq R_j^{l+1}$ (Property 3.6) and $R_i^l < R_j^l$, then we have $R_i^{l+1} < R_j^{l+1}$, which contradicts the assumption that link $j$ is removed at iteration $l+1$.

§

To further understand Lemma 3.1, let us consider the network in Figure 8. In this example, we assume that the figure is a snapshot of the remaining network at iteration $l$ of the multi-link maxmin algorithm. Notice that link 4 is "our link $j$". At iteration $l$ we have that $R_2^l, R_5^l < R_4^l$. Notice also that link 4 will be removed at iteration $l+1$, then we need to see whether links 2 and 5 satisfy the proposition. At iteration $l$, link 5 becomes a bottleneck and is removed from the network. Also, link 2 does not become a bottleneck at iteration $l$ but link 1 shares a flow with link 2 and link 1 becomes a bottleneck at iteration 2. The important meaning of this last case is that when link 1 is removed, the advertised rate at the following iteration for link 2 increases beyond that of link 4, allowing the latter to be the new bottleneck.
The previous property provides the means to understand the order of link convergence. In the previous example and using our ordering definition, we can say that “link 4 cannot converge until link 1 and link 5 have converged”, or more formally, link 1 \( \preceq_m \) link 4 and link 5 \( \preceq_m \) link 4. Note also that link 4 can converge without the need for link 2 to converge. The following definition and corollary formalize this result.

**Definition 3.14. Direct/indirect precedent link and medium Link.** Let \( N \) be a network configuration and let link \( j \) be a link in \( N \) removed at level \( l+1 \). Let also \( i \) be a link in \( N \) that at iteration \( l \) shares a flow with link \( j \) so that \( R_i^l < R_j^l \). We say that link \( i \) is a **direct precedent link of link** \( j \) if it is removed at level \( l \). On the other hand, if link \( i \) is not removed at level \( l+1 \), then let link \( k \) be a link that shares at least one flow with link \( i \) and that is removed at level \( l \). Note that by Lemma 3.1 such a link exists. We call link \( k \) an **indirect precedent link of link** \( j \) and link \( i \) is referred as the **medium link of links** \( j \) and \( k \).

Continuing in our example of Figure 8, we have that link 5 is a direct precedent of link 4, link 1 is an indirect precedent of link 4 and link 2 is the medium link of links 1 and 4.
Corollary 3.1. A link cannot converge to its maxmin rate until all of its direct and indirect precedence links have converged.

6.3 CPG Graph

Definition 3.15. Constrained Precedence Graph (CPG). Let \( N \) be a network configuration and let \( j \) and \( k \) be any two links in \( N \). We define the Constrained Precedence Graph (CPG) as the directed graph that is built using the following rules:

- Each node represents a link that is removed in the multi-link maxmin algorithm.

- An arc runs from \( k \) to \( j \) if and only if \( k \) is a direct or indirect precedent link of link \( j \).

Let us illustrate the concept of CPG graph through an example. Consider the network example in Figure 9. The corresponding CPG graph is shown in Figure 10.

![Figure 9 Network example](image)

At the first iteration, link 1 is the only link that is removed with an advertised rate (AR) of 10. Note for example that link 6 cannot be removed because link 3 has a smaller AR, and at the same time link 3 cannot be removed because link 2 has a smaller AR. At the second iteration, link 2 is removed with an AR of 15 and link 1 becomes its direct precedence. Again, link 6 cannot be removed because link 3 has a smaller AR. However,
at iteration 3 we note that the AR of link 3 becomes bigger than that of link 6. At this iteration, link 6 is removed with no direct precedence link. Instead, it has link 2 as an indirect precedence link and link 3 as the medium link for such relation. Finally, at iteration 4 links 8 and 9 can be removed in parallel since they don’t share a flow.

![Figure 10 CPG graph](image)

The CPG graph provides two types of information. First, it gives the ordering of convergence for each link. For instance, in the previous example we discovered that even if link 2 and link 6 don’t share a common flow, they are related in the sense that link 6 can converge only if link 2 has converged. The CPG graph also gives a lower bound in the convergence time of a maxmin algorithm.

*Corollary 3.2.* Let $i$ and $j$ be two links in a network $N$. Link $i <_m$ Link $j$ if and only if there exists a directed path in the CPG graph from link $i$ to link $j$.

*Corollary 3.3. Maxmin convergence time.* The convergence time of a maxmin algorithm is at least $(L-1)T$, where $L$ is the number of levels in the CPG graph (e.g. in Figure 10 we have $L = 4$) and $T$ is the time required for a link to converge once all of its predecessor links have converged.
Corollary 3.3 applies to both centralized and distributed algorithms. In centralized algorithms, \( T \) is the computational time spent in a single iteration. In distributed algorithms, \( T \) is the time that it takes for a link to receive the status from its predecessors links.

### 6.4 CPG Algorithm

We now focus on the construction of the CPG graph. The algorithm presented in this section is an extension of the centralized CPG algorithm in Figure 6. In order to understand the procedure, let us first define some notation.

**Definition 3.16. Set of potential direct precedence links.** Let \( N \) be a network and let \( i \) be one of its links. At iteration \( l \) of the centralized algorithm, we define the set of potential direct precedence links of link \( i \), denoted by \( \Delta_i^l \), as the set of links such that:

- If link \( j \) belongs to \( \Delta_i^l \), then link \( i \) and link \( j \) share a flow,

- If link \( j \) belongs to \( \Delta_i^l \), then \( R_j^l < R_i^l \) and

- If link \( j \) belongs to \( \Delta_i^l \), then link \( j \) is removed from the network at iteration \( l \).

**Definition 3.17. Set of Potential indirect precedence links.** Let \( N \) be a network and let \( i \) be one of its links. At iteration \( l \) of the centralized algorithm, we define the set of potential indirect precedence links of link \( i \), denoted by \( I_i^l \), as the set of links such that:

- If link \( j \) belongs to \( I_i^l \), then link \( j \) and link \( i \) do not share a flow and there exists a link, call it link \( k \), that shares at least one flow with link \( j \) and link \( i \),
- If link \( j \) belongs to \( I_i^j \), then \( R_k^i < R_j^i \) and

- If link \( j \) belongs to \( I_i^j \), then link \( k \) is not removed from the network at iteration \( l \) and link \( j \) is removed from the network at iteration \( l \).

By construction of our theory, the following two corollaries hold.

**Corollary 3.4.** The set of direct precedence links of link \( i \) is \( \Delta_i^{l-1} \), where \( l \) is the iteration of the centralized algorithm at which link \( i \) is removed.

**Corollary 3.5.** The set of indirect precedence links of link \( i \) is \( I_i^{l-1} \), where \( l \) is the iteration of the centralized algorithm at which link \( i \) is removed.

The previous two corollaries provide the means to build the CPG graph. In the algorithm, the set of potential direct and indirect precedence links are stored in a per-link basis and recomputed at each iteration. If a link is removed, its current set of potential direct/indirect precedence link becomes its actual set of direct/indirect precedence links and we add them into the CPG graph. The algorithm is presented in Figure 11.

**BuildCPGGraph (N)**

---

**Input parameters:**

- \( N \): a network with a single link

**Output parameters:**

- \( G \): the CPG graph of network \( N \)

1. \( L = 1; \Delta_i^0 = \{\varnothing\} \) and \( I_i^0 = \{\varnothing\} \) for every link \( i \) in the network \( N \);

2. For each single link \( N_j \) in \( N \), obtain \( R_j \) and \( \{r_i \mid i \in V_j\} \) by executing \( \text{SolveSingleLinkCMM}(N_j) \);

3. For each link \( j \) in \( N \) such that \( \Delta_j = \min\{R_i \mid \text{link} \ j \ & \text{link} \ k \ \text{share a joint flow}\} \), do the following:
   3.1. Update \( C_k \) for all link \( k \) sharing a joint flow with link \( j \) doing the following
$$C_i = C_i - \sum_{j \in V_i \cap \Phi_i} r_j;$$

3.2. Remove link $j$ and all the flows crossing link $j$ from $N$;

3.3. Add link $j$ to the CPG graph as a node and add a directed edge between any link in $\Delta_j^{l-1} \cup \Xi_j^{l-1}$ and link $j$.

4. If $N$ is not empty compute $\Delta_i^l$ and $\Xi_i^l$ for every remaining link $i$, do $L = L + 1$ and go to step 2. Otherwise, stop.

Figure 11 BuildCPGGraph algorithm
Chapter IV

DISTRIBUTED UNICAST MAXMIN PROTOCOL

“A more egalitarian society, with no differential payoff to effort and to ability, however acquired, might well be a more serene society”

Paul A. Samuelson (1970)

1 Introduction

A flow control protocol is a mechanism that allows the sources to adapt their rates according to some feedback received from the network. Depending on the nature of this feedback, flow control protocols can be classified into two groups: explicit or implicit feedback protocols. In implicit feedback schemes, the source infers a change in its service rate by measuring its current performance. In explicit feedback schemes, the information is explicitly conveyed to the source. Among the explicit flow control approaches, the so-called maxmin allocation has been widely adopted as a solution of the rate assignment problem. Several authors have approached the flow control problem from the maxmin perspective. [CHA95] and [TSA96] first addressed the classic maxmin problem. Later,
[HOU98], [LON99], [ABR97] and [KAL97] studied the maxmin problem with additional non-linear constraints, the so-called maxmin with minimal rate guarantee.

The common denominator in the above references is that they are all state maintained algorithms, that is, per flow information is maintained at each switch. Other authors have approached the problem using stateless algorithms. Stateless schemes such as ERICA [JAI96] and EPRCA [ROB94] are good in the sense that they minimize the computational cost in the switch giving a higher degree of scalability. The cons of this approach are higher convergence time and in some cases failure to guarantee fairness for some scenarios, degrading the level of QoS in the network.

2 Stateless or State Maintained Protocols

Because of the exponential growth of the Internet, one of the important properties that a protocol has to consider is scalability. Many of the explicit rate flow control algorithms were first applied to ATM networks. Originally, the ATM model was based on the end-to-end virtual circuit (VC) model, in which users trying to communicate have to set up a fixed path, the VC, before transmission. As a result of this model, the number of VCs in an ATM switch can potentially be very high, so high that any additional complexity in the switch on a per VC basis can be unaffordable. Under this model, any state maintained algorithm can be very expensive and stateless algorithms may probably be the only affordable solution.

The network model has shifted since. Further scalability issues have forced the way networks are built. For instance, consider the Multiprotocol Label Switching model
An MPLS network can be seen as a scalable version of an ATM network. To provide scalability, an MPLS network implements a higher level of flow granularity. A flow (equivalent to the concept of VC in the ATM notation) aggregates many sub-flows with similar properties such as routing path or quality of service requirements. By making use of this larger granularity, the number of flows to be handled in a router is decreased, so much that per flow computation is now in most cases affordable.

The protocol presented in this paper assumes a connection-oriented network with scalability properties such as those of an MPLS network. Under this assumption we will argue that the performance improvement achieved by using a state maintained versus a stateless protocol is significant.

3 Distributed Protocol

A distributed algorithm differs from a centralized algorithm in the sense that input parameters of the algorithm are not positioned in a single location. In order to converge to the same solution as in the centralized approach, distributed algorithms need to provide a transport support to distribute the information. Information has to be carried to the right place so that timely and precise decisions can be made.

In our distributed solution, decisions are taken in the switches. We will provide a signaling protocol that allows the switches to virtually establish a one to one communication link between them. Intuitively, once the right information has been received, the switch can deduce whether it is a bottleneck or not. If it is a bottleneck, it
will inform the other switches so that they can proceed with their computations. Once a switch has converged to the optimal rate allocation, a source is also immediately informed.

The following sections present the algorithm in two steps. First, we will explain the signaling algorithm that transports the information from switch to switch and from switch to source. Then, we will explain the available rate computation algorithm that allows the switch to know whether it is a bottleneck or not.

3.1 Signaling Protocol

The signaling scheme proposed in this section uses the framework defined in the ATM Traffic Management Specifications 4.0 [ATM96]. In our signaling protocol we will assume that special resource management packets (RM packets) are periodically sent from source to destination (downstream direction) and then back to the source (upstream direction). The former are called forward RM (FRM) packets and the latter are called backward RM (BRM) packets. RM packets include a field called explicit rate (ER). These packets travel along the source-destination path capturing in ER the value of the available bandwidth in the bottleneck switch. After convergence, the value of this field for a backward RM packet reaching the source should be equal to the optimal transmission rate of this source.

Note that if RM packets are not available in the network, the ER field can be piggybacked in a data packet. The use of RM packets is preferable though since it allows the assignment of different levels of priorities between data packets and network management packets. In a situation of congestion, network management packets should
have the highest priority since they provide the means to remove the congestion. The simile for this approach is a police automobile. Policemen drive special cars and use the siren to get the highest priority in the highways. In our case, the siren metaphor can be implemented by inserting an s-bit field in the RM packet. A switch that is congested can set this bit so that the packet is assigned the highest priority in the network.

Figure 1 shows the source algorithm. Every time a backward RM packet arrives, the transmission rate of the source (TR) is set to the ER field in the RM packet. In addition, a source periodically sends forward RM packets. The initial values of the ER field and the s-bit in this packet are set to infinity and zero, respectively.

SourceAlgorithm ()
When a RM packet arrives
   TR ← RM.ER;
When is time to send a RM packet
   RM.ER ← ∞;
   RM.S ← 0;
   Send RM packet downstream;

Figure 1 Source algorithm

Figure 2 shows the destination algorithm. Upon receiving a forward RM packet, the ER field is set to infinity and the packet is sent back to the source. This implementation differs from previous approaches where the destination does not modify the ER value. As we will see, resetting ER to infinity in the destination site is crucial to achieve switch-to-switch direct communication and fast convergence.

DestinationAlgorithm ()
When a RM packet arrives
   RM.ER ← ∞;
Before presenting the switch algorithm, Figure 3 introduces the flow architecture of the switch. In this configuration, the switch has four input ports (i-ports) and four output ports (o-ports). We assume that data flows from left to right. There are currently two VCs flowing through the switch. Flow 1 arrives from i-port 1 and departs from o-port 4. Hence, this VC contends for link resources C₁ and C₈. Similarly, flow 2 arrives from i-port 4 and departs from o-port 4, contending for link resources C₄ and C₈. In this example, the two flows will interact when contending for resources in o-port 4, that is C₈. Such is the type of contention that our maxmin protocol will resolve.

Figure 4 shows the switch algorithm. The switch stores some status for each flow. The meaning of these fields is: UB for upstream available bandwidth, DB for downstream...
available bandwidth, \( N \) for the number of flows crossing the switch, \( MFR \) for minimal flow rate and \( B \) for the minimal of \( UB \) and \( DB \).

\[ \text{SwitchAlgorithm()} \]

\textbf{When a connection flow} \( i \) \textbf{is set up}

- Allocate new entries for \( UB_i \), \( DB_i \), \( MFR_i \), and \( B_i \);
- Set \( MFR_i \) to the minimal rate allowed by the session;
- \( N \leftarrow N + 1 \);

\textbf{When a connection is closed}

- Free memory space reserved for the connection;
- \( N \leftarrow N - 1 \);

\textbf{When a new RM packet arrives from flow} \( i \)

- If forward RM packet
  - \( UB_i \leftarrow RM.ER \);
- Else
  - \( DB_i \leftarrow RM.ER \);
  - \( B_i \leftarrow \min(UB_i, DB_i) \);
  - ComputeAR ()
  - \( RM.ER \leftarrow \max\{\min\{AR, B_i\}, MFR_i\} \);

\textbf{If the switch is congested}

- \( RM.S \leftarrow 1 \);

\textbf{If} \( RM.S = 1 \)

- Forward RM packet with maximum priority;

\textbf{Else}

- Forward RM packet;

\[ \text{Figure 4 Switch algorithm} \]

When a connection is setup - for example in a MPLS network this would correspond to the setup phase in the LDP (Label Distribution Protocol, see [MPL00]) - we allocate memory space for the parameters, store the value of the minimal rate for a flow (\( MFR \)) and increase the number of flows crossing the switch. When a connection is closed, we free the memory space corresponding to the parameters and decrease the number of flows.
The actual signaling algorithm is executed every time an RM packet arrives. If it is a forward RM packet we save the ER field in UB, otherwise we save it in DB. This part is also crucial to achieve switch-to-switch communication. Intuitively, flow and switch wise, the status of the whole network can be summarized using two pieces of information: the upstream and the downstream available bandwidth. This concept will be further explained at the end of this subsection.

The switch algorithm proceeds by computing its own available rate calling \textit{ComputeAR}. This computation will depend on B, the minimal value of UB and DB. Another property worth noticing is that we can cleanly separate the signaling algorithm from the rate computation algorithm. One can change the ComputeAR procedure to accommodate a different optimization criterion and still use the same signaling protocol to keep switch-to-switch communication properties.

Finally, the switch updates the ER field in the RM packet to the maximum of the MFR and the minimum of the switch available rate and the current ER value. After treating the s-bit properly, we forward the packet.

Before defining the ComputeAR algorithm, let us first understand why this signaling protocol allows for a fast convergence implementation. The proposed protocol has two properties: \textit{bi-directional minimization} and \textit{transient oscillation freedom}.

\textit{Property 4.1. Bi-directional minimization.} Most of previous state maintained algorithms [CHA95-HOU98-LON99-KAL97] do not reset the ER field to infinity when returning RM packets at the destination site. Also, such is the approach taken by the ATM Traffic Management Specifications [ATM96]. If this was the case, let us assume the network
snapshot shown in Figure 5. In steady state, switch 2 is receiving backward RM packets with ER equal to 5 (since the destination does not reset this value to infinity). Now suppose that $B_1$ increases to 15 so that the new bottleneck is 10. Then, for switch 2 to receive this new bottleneck value it will have to wait for an RM packet to go from switch 1 to destination and then back to switch 2. In other words, the signaling protocol does not provide the necessary means for switch 3 and switch 2 to directly talk.

![Network example](image)

*Figure 5 Network example*

In our approach, for switch 2 to receive the new bottleneck value we will only have to wait for the next RM packet coming from switch 3. Information is not conveyed in a round trip manner but in a one-way trip. Therefore, a virtual switch-to-switch direct communication is achieved that leads to a convergence time twice as fast.

**Property 4.2. Transient oscillation freedom.** Another important protocol implementation issue is the storage of both the upstream and downstream available bandwidths in different fields. To illustrate this concept, we will also use the same example shown in Figure 5. Suppose that switch 2 does not record the values of UB and DB. In that case, after switch 1 increases its available bandwidth to 15 it will inform switch 2. However, switch 2 cannot make a good decision since it does not recall that the downstream path is bottlenecked at a rate of 10. If the switch decides that the new bottleneck rate is 15, the new situation can be dangerous since this faulty value can propagate to other flows in the network inducing oscillations and new congestion spots (Figure 6). Instead, if switch 2
remembers the downstream available bandwidth, upon receiving the feedback from switch 1 it can immediately recognize that switch 3 is the new bottleneck.

Figure 6 A wrong ER value of 15 is advertised, the correct ER value should be 10
3.2 Rate Computation Algorithm

In our approach, each flow $i$ has a minimal flow rate $MFR_i$ that must be guaranteed. From a switch standpoint, each flow $i$ also has a peak flow rate equal to $B_j$, the minimum of $UB_i$ and $DB_i$. Indeed, note that a switch must not give more bandwidth to a flow $i$ than $B_j$ since there must be another switch that cannot afford more than such amount of bandwidth. As a result, a switch must solve a single link maxmin problem with both peak and minimum flow rate constraints (Figure 7a). In order to solve this problem, we first transform it into an equivalent one: a multi-link problem with no peak flow rate constraints. For that, we use the concept of extended network (Definition 3.2). As shown in Figure 7b, each peak flow rate $i$ is substituted by a new link connected to a switch with capacity equal to $B_j$. From Property 3.2, the available rate in both networks is the same.

![Figure 7 Network transformation](image)

Note that now our problem is that of solving a multi-link network where we know all of its parameters (Figure 7b). Therefore, we can use the centralized algorithm to solve it. Figure 8 presents an implementation of the ComputeAR function. The reader can check
that this implementation is the same as that presented in Chapter III-Figure 6 but for the case of our particular network in Figure 5b. Because the algorithm requires two loops, its cost is $O(N^2)$.

\begin{verbatim}
ComputeAR()

Parameters:
\begin{itemize}
  \item $\Omega$: Set of rates bottlenecked at their MFRs;
  \item $\Psi$: Set of rates bottlenecked somewhere else;
  \item $RC$: Remaining Capacity;
  \item $RN$: Remaining flows that are not in $\Omega \cup \Psi$
  \item $AR$: Advertised Rate;
\end{itemize}

Input: 
The current state of the switch

Output: 
The new advertised rate

1. $\Psi \leftarrow \emptyset; RC \leftarrow C; RN \leftarrow N$;
2. $\Omega \leftarrow \emptyset$;
3. $AR \leftarrow RC/RN$;
4. If $\exists i \notin (\Omega \cup \Psi)$ such that $AR < MCR_i$,
   4.1. Put any flow $i$ such that $i \notin (\Omega \cup \Psi)$ and $AR < MCR_i$ into $\Omega$;
   4.2. $RC \leftarrow C - \sum_{i \in \Omega} MCR_i - \sum_{i \in \Psi} B_i; RN \leftarrow N - |\Omega \cup \Psi|$;
   4.3. Return to 3;
5. If $\exists i \notin (\Omega \cup \Psi)$ such that $AR > B_i$,
   5.1. Put any flow $i$ such that $i \notin (\Omega \cup \Psi)$ and $AR > B_i$ into $\Psi$;
   5.2. $RC \leftarrow C - \sum_{i \in \Psi} B_i; RN \leftarrow N - |\Psi|$;
   5.3. Return to 2;
6. Stop;
\end{verbatim}

\textit{Figure 8 ComputeAR Procedure to solve the single-link case}
3.3 Protocol Convergence

In this section, we will prove the convergence and the time complexity of the presented protocol. We first introduce a preliminary definition of convergence time that later we will use to measure our protocol efficiency.

Definition 4.1. Convergence time definition. Let \( t_0 \) be the time when the link capacities have stabilized in the network and let \( t_i \) be a time when the sources have converged to their maxmin allocation. We define the convergence time of the distributed algorithm as
\[
\min_j \{ t_i - t_0 \}.
\]

Let us now study the convergence properties of the distributed protocol.

Lemma 4.1. Link convergence condition. Let \( AR_j^1 \) be the advertised rate computed at the first iteration of the CPG centralized algorithm (Chapter III-Figure 6) for an arbitrary link \( j \). Then, given any arbitrary state of this link, that is, given arbitrary values of the fields \( B_i \), the rate computed by the procedure ComputeAR is always greater or equal than \( AR_j^1 \).

Proof. Note first that \( AR_j^1 \) can be obtained by setting \( B_i \) to infinity, for all \( i \), and then executing ComputeAR. This must be true if we notice that at the first iteration of the centralized CPG algorithm all the links assume that all their crossing flows are not constrained elsewhere. It is obvious that the first time the algorithm executes step 5 of ComputeAR, the value of AR is equal to \( AR_j^1 \), since at that time the values of \( B_i \) haven’t been considered yet. Now, considering the monotonic increment behavior of AR
(Property 3.6) and recalling that ComputeAR is equivalent to the CPG algorithm applied to the network configuration in Figure 7b, we have that the AR value increases or stays the same at each iteration of the ComputeAR. Then the lemma must hold.

Theorem 4.1. Network Convergence. Given any arbitrary initial conditions on the state of the links and the state of the RM packets in transit, the proposed distributed algorithm converges to the maxmin rates as long as the number of established flows and the available bandwidth at each switch eventually stabilize.

Proof. Let $t_0$ be the time at which the set of sessions and the available bandwidth stabilize. From lemma 4.1 we know that the AR value computed independently at an arbitrary link $j$ is greater or equal than the advertised rate computed at the first level of the centralized algorithm for link $j$, $AR_j^1$. Let $D$ be an upper bound on the one-way delay (half the round trip delay) in the whole network. Note that because the algorithm updates RM packets in both forward and backward ways, the time required for a link to know about the change of the state of another link, both sharing a flow, is at most $D$. Let link $k$ be an arbitrary link belonging to the first CPG level. From the centralized algorithm we have that $AR_j^1 \geq AR_k^1$ for any link $j$ that shares a flow with link $k$. Then, at time $t_0 + D$ we have that the state of link $k$ satisfies $B_i \geq AR_k^1$ for any $i$. Now when executing algorithm ComputeAR at link $k$ we have that the first time that step 5 is executed its checking condition will not hold and the algorithm will finish returning $AR = AR_k^1$. Then, at time $t_0 + D$ all the first level bottleneck links have converged to their maxmin
advertised rate. This means that at time $t_0 + 2D$ any link will be aware of the fact that the first level bottlenecks have converged. Moreover, since $AR_j^1 \geq AR_k^1$ for any link $j$ that shares a flow with link $k$, we have that at any link the value of a $B_i$ field for a flow $i$ bottlenecked at the first level will be set to its maxmin rate and will not be changed anymore. In other words, every time after $t_0 + 2D$, when a link $j$ which is not a first level link computes AR it will find that this set of first level flows are constrained somewhere else (in a first level bottleneck). So we can remove this flows from the network and subtract their assigned rate to the links they cross. Now, applying again lemma 4.1, we know that the AR value computed independently at an arbitrary link $j$ is greater or equal than the advertised rate computed at the second level of the centralized algorithm for link $j$, $AR_j^2$, which means that we are in the same conditions as we were at the beginning of this proof but one iteration further. Hence, we can use the same argument we applied in the first level step to prove that the second level bottleneck links converge to their maxmin solution as well. Finally, by induction we prove that theorem 4.1 must hold.

Let us now illustrate with an example how the ER packets carrying bottleneck information travel along the network from link to link. We consider a network configuration with four CPG levels. Figure 9 shows six snapshots of the network while converging to the maxmin solution. The links in the network are grouped into sets of links belonging to the same CPG level. The thin arrows between the groups show the flow of ER packets that bring the relevant information for maxmin convergence. At time $t_0 + D$ all the first level links have received enough information to converge. Once the
first level links have converged, they let the rest of the network know about their new status. At $t_0 + 2D$ every link in the network reestablishes its status according to the fact that first level links have converged. With that information, $D$ units of time later, the second level links converge to their maxmin rates after receiving ER packets from the third and the four level links. The process is repeated until all the four level links have converged.

**Theorem 4.2. Convergence Time.** Let $N$ be a network with up to $L$ CPG levels and let $t_0$ be the time at which the set of sessions and the available bandwidth stabilize. Then any $L$-level link will have converged to the maxmin solution after $t_0 + 2(L-1) \cdot D$. In terms of the round trip delay $RTT$, that is $t_0 + (L-1) \cdot RTT$.

**Proof.** From the proof of theorem 4.1 we know that for a network with up to $L$ CPG levels the following must be true:

- It takes $D$ units of time for the 1-level links to converge.

- It takes $(2 \cdot i - 1) \cdot D$ units of time for the $i$-level links to converge.

- It takes $2(L-1) \cdot D$ units of time for the $L$-level links to converge.

In terms of round trip time, we have that it takes $(L-1) \cdot RTT$ units of time for the $L$-level links to converge.
Per our knowledge, the convergence time provided in Theorem 4.2 is the lowest for a state maintained distributed maxmin algorithm with minimal rate guarantees. For a comparison, table 1 shows the convergence time of previous approaches given by their authors. In this table, $N$ denotes the number of bottleneck links, $S$ denotes the number of flows and $L$ denotes the number of CPG levels. Note that $L \leq N \leq S$.

![Figure 9 Convergence steps](image)

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Convergence time</td>
<td>$4(N-1)RTT$</td>
<td>$2.5(S-1)RTT$</td>
<td>$2(L-1)RTT$</td>
<td>$(L-1)RTT$</td>
</tr>
</tbody>
</table>

*Table 1 Convergence times*
The table above shows that our approach improves the convergence time by a factor of 2. As we have already stated, this improvement is achieved by using a bi-directional minimization scheme and by maintaining switch state for both upstream and downstream bottleneck rates. In general, the performance of a distributed maxmin algorithm depends on two factors: the efficiency in the algorithm used to bring information from one switch to another and the efficiency in the algorithm executed every time a packet arrives in a switch. In this section, we have discussed the complexity of the first factor. For the second factor, in Figure 8 we presented the procedure ComputeAR that incurs a per-packet processing complexity of $O(N^2)$, $N$ being the number of flows crossing the switch. In the following section, we introduce an alternative algorithm that significantly reduces the complexity of the ComputeAR algorithm.

3.4 Fluid Model: the Maxmin Vessel

In the following discussion we will assume without loss of generality that $MFR_1 \leq MFR_2 \leq \ldots \leq MFR_N$. Let us consider the vessel in Figure 10a. It consists of a rectangular cavity with steps in both the lower and the upper side of it. The lower side step $i$ is defined at a height equal to $MFR_i - MFR_i$, whereas the higher side step $i$ is defined at a height equal to $B_i - MFR_i$, both in units of length.
Suppose now that we let \( C - \sum_{i=1}^{N} MFR_i \) units of volume of water flow into the vessel as shown in Figure 10b. It can be seen that there is a direct relation between the height of the resulting water level and the maxmin solution of our network in Figure 5b. First, note that we have removed all the MFRs from the amount of water so that we guarantee that each flow gets at least its minimal rate requirement. The lower steps in the vessel are built so that we first serve those flows with smaller MFR. When the first flow of water comes, we start filling the first step. If there is enough water left, then we start filling the second step at the same pace as the first one, and so on. If the level of water hits an upper
side step, then its corresponding flow is saturated and no more water (bandwidth) is given
to it.

The fluid model allows us to build and algorithm that reduces the cost of computing the
available rate in a switch. As the model suggests, this algorithm is implemented in two
steps. The first procedure is called \textit{BuildVessel} and is used to build the vessel. This
algorithm is shown in Figure 11 and its output is a $2 \times 2N$ matrix representing the vessel.
The second procedure is called \textit{FindWaterLevel} and it computes the available rate. As
shown in Figure 11, FindWaterLevel receives a vessel $d$, the amount of water $C^*$ and
returns the available rate $AR$. Finally, Figure 13 shows how these two functions are to be
coded inside the new ComputeAR procedure.

\textbf{BuildVessel()}

\textit{Input parameters:}
\begin{align*}
MCR' &= \min\{MCR_i \mid i = 1, \ldots, N\}; \\
M &= \{MCR_i - MCR' \mid i = 1, \ldots, N\}; \\
P &= \{B_i - MCR' \mid i = 1, \ldots, N\};
\end{align*}

\textit{Output parameters:}
\begin{align*}
d: \text{the vessel structure}
\end{align*}

1. Order the elements in $M \cup P$ from the smallest to the largest and put them in the first row of
the matrix $d$ of size $2 \times 2N$;
2. Build $y$ by using the following equation
\begin{align*}
y[i] &= \begin{cases} 1, & \text{if } i = 1 \\ y[i-1]+1, & \text{if } d[1,i] \in M \text{ and } i > 1 \\ y[i-1]-1, & \text{if } d[1,i] \in P \text{ and } i > 1 \end{cases}, \text{for } i = 1, \ldots, 2N;
\end{align*}
3. Build the second row of $d$ by using the following equation
\begin{align*}
d[2,i] &= \begin{cases} d[1,1], & \text{if } i = 1 \\ d[1,i-1] + y[i-1]d[1,i] - d[1,i-1], & \text{otherwise} \end{cases}, \text{for } i = 1, \ldots, 2N;
\end{align*}
3. Return(d);

---

**Figure 11 BuildVessel algorithm**

**FindWaterLevel()**

**Input parameters:**

- Vessel d;
- \( C^* = C - \sum_{i=1}^{N} MCR_i \);

1. Find i such that
   \[ d_{[2,i]} \leq C^* \leq d_{[2,i+1]} \];
2. \( AR = d_{[1,i]} + (C^* - d_{[2,i]}) \frac{d_{[1,i+1]} - d_{[1,i]}}{d_{[2,i+1]} - d_{[2,i]}} \)
3. Return (AR);

---

**Figure 12 FindWaterLevel algorithm**

**ComputeAR()**

1. **BuildVessel();**
2. **FindWaterLevel();**

---

**Figure 13 ComputeAR using the fluid model approach**

**Property 4.3. Algorithm computational cost.** The cost to build the vessel is \( O(N \log(N)) \) and the cost to compute the water level is \( O(\log(N)) \).

**Proof.** In Figure 11 step 1 is \( O(N \log(N)) \) and steps 2 and 3 are \( O(N) \). In Figure 12 step 1 is \( O(\log(N)) \) and step 3 is \( O(1) \).
In theory, both BuildVessel and FindWaterLevel functions should be called upon the arrival of an RM packet (as shown in Figure 13). However, in practice we don’t need to rebuild the complete vessel every time a new RM packet arrives. Intuitively, upon the arrival of a new RM packet from flow $i$, only the field $B_i$ may need to be modified. All the other fields $B_j$ for $j \neq i$ maintain the same value. In general, we can expect the new vessel to be correlated to the previous one. This suggests the construction of the vessel by using an update approach, instead of a build-from-scratch approach. That is, instead of rebuilding the vessel every time, we can build it the first time when the switch is being initialized and use a simpler algorithm to update the vessel every time a new feedback arrives. Figure 14 with Table 2 details this algorithm. At step 1, the new value of $B_i$ is ordered into its new position. At step 2, the levels of water are recomputed, if necessary. Figure 15 shows how to implement the ComputeAR function using the UpdateVessel procedure.

\[\text{UpdateVessel()}\]

\textbf{Input:}

\begin{itemize}
  \item \textit{Current vessel}: $d^{old}$ ;
  \item \textit{Current ER feedback from flow $i$}: $B_i^{old}$
  \item \textit{New ER feedback from flow $i$}: $B_i^{new}$
\end{itemize}

\textbf{Output:}

\begin{itemize}
  \item The new updated vessel: $d^{new}$ ;
\end{itemize}

\begin{enumerate}
  \item Find $j_{new}$ such that
  \[d^{old}[1, j_{new}] \leq B_i - MCR_i \leq d^{old}[1, j_{new} + 1] ;\]
\end{enumerate}
2. Let $j_{\text{old}}$ be such that $d_{\text{old}}[1, j_{\text{old}}] = B_{i}^{\text{old}}$. Compute a new vessel $d_{\text{new}}$ using equations in Table 2;

3. Return ($d_{\text{new}}$);

Figure 14 UpdateVessel algorithm

<table>
<thead>
<tr>
<th>$B_{i}^{\text{new}} &gt; B_{i}^{\text{old}}$</th>
<th>$k &lt; j_{\text{old}}$</th>
<th>$k = j_{\text{new}}$</th>
<th>$j_{\text{old}} \leq k &lt; j_{\text{new}}$</th>
<th>$j_{\text{new}} &lt; k$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$B_{i}^{\text{new}} &lt; B_{i}^{\text{old}}$</td>
<td>$d_{\text{new}}[\cdot, k] = d_{\text{old}}[\cdot, k]$</td>
<td>$d_{\text{new}}[1, j_{\text{new}}] = B_{i}^{\text{new}}$</td>
<td>$d_{\text{new}}[1, k] = d_{\text{old}}[1, k + 1]$</td>
<td>$d_{\text{new}}[1, k] = d_{\text{old}}[1, k]$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$d_{\text{new}}[2, k] = d_{\text{old}}[2, k + 1] + (d_{\text{old}}[1, k + 1] - B_{i}^{\text{old}})$</td>
<td>$d_{\text{new}}[2, k] = d_{\text{old}}[2, k] + B_{i}^{\text{new}} - B_{i}^{\text{old}}$</td>
</tr>
<tr>
<td>$B_{i}^{\text{new}} = B_{i}^{\text{old}}$</td>
<td>$d_{\text{new}}[\cdot, k] = d_{\text{old}}[\cdot, k]$</td>
<td>$d_{\text{new}}[1, j_{\text{new}}] = B_{i}^{\text{new}}$</td>
<td>$d_{\text{new}}[1, k] = d_{\text{old}}[1, k - 1]$</td>
<td>$d_{\text{new}}[1, k] = d_{\text{old}}[1, k]$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$d_{\text{new}}[2, k] = d_{\text{old}}[2, k - 1] - (d_{\text{old}}[1, k - 1] - B_{i}^{\text{new}})$</td>
<td>$d_{\text{new}}[2, k] = d_{\text{old}}[2, k] + B_{i}^{\text{new}} - B_{i}^{\text{old}}$</td>
</tr>
</tbody>
</table>

Table 2 Computation of a new vessel upon the arrival of a new feedback $B_{i}^{\text{new}}$

Algorithm: ComputeAR

1. If we haven’t build the vessel before, 
   BuildVessel();

5 See Notation section at the beginning of this dissertation

83
Else
    UpdateVessel();
2. FindWaterLevel();

---

Figure 15 ComputeAR using the fluid model with an update approach

Property 4.4. Update vessel cost. The cost to update the vessel is $O(\log(N))$.

Proof. The cost of step 1 is $O(\log(N))$, the cost of step 2 is $O(1)$.

§

The previous property shows that one can execute the switch algorithm incurring a total cost of $O(\log(N))$. Hence, the presented algorithm can scale for large number of flows.

---

Figure 16 Prototype 1.0 of the lexicographic vessel

4 Simulations

In this section we will evaluate the performance of the presented protocol. In order to have a reference, we have chosen to compare our algorithm with ERICA [JAI96]. Some of the performance parameters that we will consider are: convergence time, degree of fairness, degree of oscillation, and congestion in the queues.
4.1 Simulation Setup

Figure 17 shows the network setup for our simulation. It consists of 5 flows. In order to simulate the case of moving bottlenecks with dynamic available bandwidth, flow 5 is given a higher priority service than the other flows. In this configuration, a switch will only forward packets from flows 1-2-3-4 if it has no pending packets from flow 5.

4.2 Response Time

We measure the response time of our distributed algorithm and compare it to ERICA. The link capacities for L1 and L2 are set to 60 and 30 Mbps, respectively, and we add a minimal rate guarantee to flows 3 and 4 of 40 and 20 Mbps, respectively. We disable flow 5 so that available bandwidths in the links are fixed. The length of link 1 and 2 are set to 100 km, introducing each one a propagation delay of 1 millisecond. The initial transmission rates of all the flows are set to 7.5 Mbps. Note that the maxmin solution to this network configuration is $r_1 = 10$, $r_2 = 10$, $r_3 = 40$, $r_4 = 20$, where flow 3 and flow 4 are constrained at their minimal flow rate requirements.

![Figure 17 Network configuration](image)

$\text{Figure 17 Network configuration}$
Figure 18 shows the response time for both algorithms. It takes about 2.6 milliseconds for our protocol to converge to the maxmin solution and once in steady state, the rates take the exact maxmin value. ERICA is slower in terms of convergence time. It takes about 256.7 milliseconds to converge to a rate 99% close to the maxmin solution. While our distributed algorithm achieves direct convergence, ERICA converges asymptotically. This is due to the stateless characteristic of ERICA: switches doing ERICA do not keep track of the bottleneck rates of previous CPG iterations and therefore more round trip delays are required to converge. The results in this simulation prove that by adding some state in the switch, the convergence time can be improved by a factor of two orders of magnitude.

![Figure 18 Response time of (a) our distributed algorithm (b) ERICA](image)
4.3 Dynamic Convergence

In this simulation we consider the case of moving bottlenecks. For that, an on-off traffic is inserted in flow 5. The on rate is set to 100 Mbps while the off rate is set to 10 Mbps, having both duration intervals of 80 milliseconds. We reset all the minimal flow rate constraints to zero and both link capacities are set to 150 Mbps. Note that under this configuration, during an off interval flows 1, 2 and 3 are constrained at link 1 with a rate of 50 Mbps each and flow 4 is constrained at link 2 with a rate of 90 Mbps. During an on interval, flows 1 and 4 are constrained at link 2 with a rate of 25 Mbps and flow 2 and 3 are constrained at link 1 with a rate of 62.5 Mbps.

Figure 19 shows the rate allocation resulting from both our algorithm and ERICA. Our algorithm proves to converge faster and with practically no oscillation. ERICA suffers from a considerably amount of oscillations.
Figure 20 shows the queue sizes at both switch 1 and 2. From the figure, ERICA suffers some important congestion at switch 1. Note that the queue size produced by our protocol in this switch is almost negligible. In switch 2 the differences are not as relevant. However, it is worth noticing that in switch 2 the queue sizes produced by our protocol are more predictable than those produced by ERICA. The reason comes from the asymptotic convergence of ERICA. While eventually converging to the maxmin rate, ERICA will take longer time to reach optimality. During this time the rates can be considerably far from the maxmin solution, which can lead to unpredictable queue sizes. One would expect that such unpredictability would worsen as the size of the network grows.
Chapter V

MULTICAST MAXMIN THEORY

"... each person possesses an inviolability founded on justice that even the welfare of society as a whole cannot override. For this reason, justice ... does not allow that the sacrifices imposed on a few are outweighed by the larger sum of advantages enjoyed by many."  

John Rawls (1971)

1 Introduction

One could consider the communication between one source to many destinations in the same manner as that of one source to one destination. In particular, this type of communication could be achieved by establishing different unicast connections, one for each source-destination pair. The defect of such approach is that redundant packets are generated that lead to a waste of network resources (Figure 1a).

In multicast traffic, packets traverse a single tree that connects the source with the set of destinations. Instead of having a copy of each packet traversing the same link, multicast packets are only duplicated at the branching points of the tree (Figure 1b). In this
environment, there are two ways to allocate rates: single-rate or multi-rate allocation. For single-rate allocation, the source transmits at the minimum available rate in the whole tree. In this approach, a slow receiver can potentially cause the entire multicast group to suffer a slow rate. This drawback is overcome in multi-rate multicast where data can be layered encoded allowing different transmission rates at different branches of the tree. Furthermore, a branch’s rate in the tree is no longer limited by the minimum downstream branch rate, but rather the maximum downstream branch rate.

![Multicast communication diagram](image)

*Figure 1 Multicast communication*

The single-rate multicast flow control problem has been studied before [ROB94, TZE97, ZHA99, CAV96], where the network is assumed to provide feedback information to the sources. The multi-rate problem was studied by [STE96], where the network does not participate in the flow control process.

The contribution of this chapter is twofold. First, we present a theory of multicast maxmin rate allocation. Until now, no correct theory exists for the multicast multi-rate maxmin assignment with minimum rate constraint, only unicast theory has been known. Second, a practical design of a multicast maxmin distributed protocol is presented. The
protocol maintains a good performance-scalability trade-off. While the network is required to participate, the complexity added by the protocol is minimal.

2 Multicast Maxmin Theory

In what follows, a VC is a virtual connection defining a set of fixed paths made of links in which packets flow from a source to a set of destinations. The most generic topological form of a VC is that of a tree.

Definition 5.1. Let \( VVC_i^j \). We say that \( VVC_i^j \) is a virtual VC of \( VC_i \) if it consists of a set of consecutive links in \( VC_i \) in which the flow of data going through satisfies:

1) it begins at either the source of \( VC_i \) or a branching point in \( VC_i \),

2) it ends at either a destination in \( VC_i \) or a branching point in \( VC_i \) and

3) it does not cross any branching point.

In the above notation we have that \( j \in I_i \), where \( I_i \) is a set of indexes that characterize each VVC in \( VC_i \).

The name virtual VC comes from the fact that a branching point in a VC can be interpreted as a set of multiple virtual sources, one for each branch, transmitting each one at different rates. Each of these virtual sources implicitly define a new flow of data within the VC.
It is important to note that within a VVC, the rate is constant. In general, note that a multicast VC may not satisfy such property. This constitutes a fundamental reason for which the concept of VVC is needed when extending classical unicast maxmin theory to the multicast case. Note that when a multicast VC takes the unicast form, then the VVC and the unicast VC definition are the same. Therefore, the theory here presented, while being more generic, it supports the case of unicast networks.

Definition 5.2. Rate $r_i^j$. We define $r_i^j$ as the rate at which data going through $VVC_i^j$ is transmitted.

In multicast, quality of service (QoS) parameters such as minimal rate constraints are defined in a per-destination basis. To support QoS, we need to define the concept of leaf.

Definition 5.3. VVC Leaf. A VVC is said to be a VVC leaf if it is incident to a destination.

We now continue with a set of definitions that will be needed to derive the theory.

Definition 5.4. VVC Source. A VVC is said to be a VVC source if it is incident to a source.

Definition 5.5. Rate vector. Given an arbitrary ordering of the VVCs in a multicast network and given a rate assigned to these VVCs, we define the rate vector $r$ as a vector such that its $ith$ component is the rate assigned to the $ith$ VVC.
Definition 5.6. Leaf-rate vector. Given an arbitrary order of the VVC leaves in a multicast network and given a rate assigned to these leaves, we define the leaf-rate vector \( r \) as a vector such that its \( i \)th component is the rate assigned to the \( i \)th VVC leaf.

Definition 5.7. Downstream set \( DS(VC_i^j) \) and upstream set \( US(VC_i^j) \). Let \( VC_i \) be a multicast virtual connection and let \( VVC_i^j \) be one of its VVCs. Suppose that we cut the graph defined by \( VC_i \) at \( VVC_i^j \). We obtain two new graphs, one that includes the source of the multicast virtual connection and one that does not include the source. Let us keep the graph that does not include the source and let us remove from it all the links that are included in \( VVC_i^j \). The remaining graph defines a set of links that we refer as the downstream set of \( VVC_i^j \), \( DS(VC_i^j) \).

Now let us consider the set of links that includes the minimum number of links that connect the source of \( VC_i \) with \( VVC_i^j \), including \( VVC_i^j \). We call such set the upstream set of \( VVC_i^j \), \( US(VC_i^j) \).

Figure 2 shows an example to illustrate the previous definitions. Note that in this figure we have that \( r_i^1 = r_i^2 = r_i^5 = 10, r_i^3 = 8, r_i^4 = 5 \).

Definition 5.8. Feasibility. Let \( V_u \) be the set of VVCs crossing link \( u \). A set of rates \( \{r_i^j\} \) is feasible if all of the following are true,

\[
F_u = \sum_{VVC_i^j \in V_u} r_i^j \leq C_u \quad \forall \text{ link } u \quad (5.1)
\]

\[
r_i^j = \max \{ r_i^k \mid VVC_i^k \in DS(VVC_i^j) \} \quad \forall \text{ VVC}_i^j \quad (5.2)
\]
where \( m_i^j \) is the minimal flow rate constraint (MFR) of the destination adjacent to leaf \( VVC_i^j \).

**Definition 5.9. Lexicographically maximal leaf-rate vector.** Let \( \bar{x} \) be the increasing permutation of vector \( x \) (Definition 2.7). A feasible leaf-rate vector \( r \) is a lexicographically maximal leaf-rate vector if for any other feasible leaf-rate vector \( r' \), \( \bar{r} \) is lexicographically greater or equal than \( \bar{r'} \).

Note that in contrast with the unicast problem studied in Chapter III, in the multicast case the optimization criterion applies to leaves rather than VCs. The new definition imposes that maxmin fairness must hold at the destination sites, because the fundamental objective in our network optimization problem is not to know how much a source can transmit but how much a destination can receive. Note also that for the particular case where a multicast tree consists of only one branch, i.e. the case of a unicast connection,
leaf is equivalent to VC and both multicast and unicast maxmin definitions are equivalent. Let us now introduce the maxmin definition for multicast networks.

**Definition 5.10. Maxmin leaf rate vector (maxmin rate vector).** A leaf rate vector $r$ (rate vector $r$) is the **maxmin leaf rate vector (maxmin rate vector)** if it is feasible and for each leaf $VVC_j^i$, $r_j^i$ cannot be increased while maintaining feasibility without decreasing $r_j^i$ for some other leaf $VVC_j^i'$ for which $r_j^i' \leq r_j^i$.

**Property 5.1. Maxmin Optimality Condition.** Given the feasibility set in Definition 5.8, the lexicographically maximal leaf rate vector (lexicographically maximal rate vector) and the maxmin leaf rate vector (maxmin rate vector) are the same.

**Proof.** By using Lemma 2.2, this property must be true if we notice that the CMMM problem can be written in terms of a LOP $(\Gamma, X)$ where $X$ is compact and convex and $\Gamma$ is the identity mapping $\Gamma x = x$.

We continue to build our multicast framework with some more definitions.

**Definition 5.11. Extended Minimal Rate Constraint.** Given an arbitrary $VVC_j^i$, we define its minimal rate constraint as,

$$m_i^{*,j} = \begin{cases} m_i^j, & \text{if } VVC_j^i \text{ is a leaf} \\ \max\{m_i^{*,j'} \mid VVC_j^{i'} \in DS(VVC_j^i)\}, & \text{otherwise} \end{cases} \quad (5.4)$$
Intuitively, the previous expression is a consequence of conditions 2 and 3 of the feasibility definition (definition 5.8), which imply the minimal rate requirement at \( VVC_i^j \): \( m_i^{*j} \leq r_i^j \).

**Definition 5.12. Advertised Rate.** Let \( r \) be a rate vector and let \( V_u \) denote the set of VVCs crossing link \( u \). Let also \( Vm_u \) be the set of VVCs in \( V_u \) such that \( m_i^{*j} = r_i^j \) at link \( u \). The quantity \( R_u \) defined below is called the *advertised rate* for link \( u \),

\[
R_u = \begin{cases} 
\max\{r_i^j|VVC_i^j \in V_u, r_i^j > m_i^{*j}\} & \text{if } Vm_u \neq V_u, F_u = C_u \\
\infty & \text{if } F_u < C_u \\
0 & \text{if } Vm_u = V_u, F_u = C_u 
\end{cases} \tag{5.5}
\]

**Definition 5.13. Constant Path of a leaf \( p_i^j \).** We define the *constant path* of a leaf \( VVC_i^j \) as:

\[
p_i^j = \{VVC_k^k | VVC_k^k \in US(VVC_i^j), r_k^k = r_i^j\} \tag{5.6}
\]

In other words, the constant path of a leaf is the set of its upstream VVCs that transport packets at the same transmission rate as that rate of the leaf.

**Definition 5.14. Bottleneck Link Constrained (BLC).** A leaf \( VVC_i^j \) is said to be *bottleneck link constrained (BLC)* if there exists at least one link \( u \in VVC_i^j \) for some \( VVC_i^k \in P_i^j \) such that,

1. \( F_u = C_u \),
2. \( r_i^j = r_i^k > m_i^{*k} \) and

96
3. \( r_i^j = R_u \).

**Definition 5.15. Minimal Rate Constrained (MRC).** A leaf \( VVC_i^j \) is said to be minimal rate constrained (MRC) if there exists at least one link \( u \in VVC_i^k \) for some \( VVC_i^k \in P_i^j \) such that,

1. \( F_u = C_u \),
2. \( r_i^j = r_i^k = m_i^{\ast k} \) and
3. \( r_i^j \geq R_u \).

**Definition 5.16. Constrained and Unconstrained leaves.** A leaf \( VVC_i^j \) crossing link \( u \) is said to be a constrained leaf at link \( u \) if,

1. \( VVC_i^j \) is BLC at link \( v \neq u \), or
2. \( VVC_i^j \) is MRC at some link.

Conversely, a leaf \( VVC_i^j \) crossing link \( u \) is said to be an unconstrained leaf at link \( u \) if it is not constrained at this link.

The previous definition arises from the fact that \( VVC_i^j \) is unconstrained from link \( u \)'s point of view because \( VVC_i^j \) is not constrained by some other means to set its rate lower or higher than its own advertised rate.
Definition 5.17. Saturation. A link $u$ is said to be saturated if the total flow is equal to its capacity, that is $F_u = C_u$.

Definition 5.18. Pseudo-saturated. A link $u$ is said to be pseudo-saturated if every VVC crossing this link is constrained at it and its capacity is not fully utilized, that is $F_u < C_u$.

Let us now introduce the first optimality condition.

Theorem 5.1. Multicast Bottleneck Optimality Condition. A Feasible rate vector $r$ is maxmin if and only if every leaf $VVC_i^j$ is either BLC or MRC.

Proof: (only if part) Assume that $r$ is the maxmin rate vector and suppose there exists a leaf $VVC_i^j$ that is neither BLC nor MRC. Consider also an arbitrary link $u$ in $P_i^j$ and let $VVC_i^k$ cross this link $u$, that is $VVC_i^k \in V_u$. Note that $r_i^k = r_i^j$, since both VVCs belong to the same constant path. There are two cases to consider: (1) link $u$ is saturated or (2) link $u$ is not saturated.

Consider case (1) where link $u$ is saturated. There are two other cases, (1.1) $r_i^j = m_i^{*k}$ or (1.2) $r_i^j > m_i^{*k}$, since feasibility must hold. Assuming case (1.1) and because $r_i^j$ is not MRC, it must be that $r_i^j < R_u$. Consider two additional cases: (1.1.1) $Vm_u \neq V_u$ or (1.1.2) $Vm_u = V_u$. If $Vm_u \neq V_u$, then because of the saturation assumption and Definition 5.12, there exists a $VVC_i^{k'}$ crossing link $u$ such that $r_i^{k'} > m_i^{*k'}$ and $r_i^{k'} = R_u$. Since $r_i^j < R_u$, we have $r_i^{k'} > r_i^j$. Now let $VVC_i^{j'} \in DS(VVC_i^{k'})$ be the leaf that satisfies $r_i^{j'} = r_i^{k'}$ (note that from condition 2 in definition 5.8 such a leaf must exist). We conclude that we can
increase $r^j_i$ while maintaining feasibility at link $u$ by decreasing some other leaf's rate $r^j_{i'}$, where $r^j_{i'} > r^j_i$. Assume now case (1.1.2), where $V^m_u = V_u$. Then, from saturation condition and Definition 5.12 we have that $R_u = 0$. Since $r^j_i < R_u$ we arrive at a contradiction. Consider case (1.2). Then we can ensure that $r^j_i < R_u$. This comes from the fact that Definition 5.12 implies $r^j_i \leq R_u$ and since $V^C_i$ is not BLC, it must also be true that $r^j_i \neq R_u$. Hence, from Definition 5.12, there must exist a $V^C_i$ crossing link $u$ such that $r^{k',i} > m^{k'}_i$ and $r^{k',i} = R_u$. Since $r^j_i < R_u$, we have $r^{k',i} > r^j_i$. Now let $V^C_i \in DS(V^C_i)$ be the leaf that satisfies $r^{j',i} = r^{k',i}$ (note that as in the previous case such leaf must exist). Hence, we conclude that we can increase $r^j_i$ while maintaining feasibility at link $u$ by decreasing some other leaf's rate $r^j_{i'}$, where $r^j_{i'} > r^j_i$.

Finally, assume case (2) where link $u$ is not saturated. Then $V^C_i$ can be increased at this link while maintaining feasibility without dropping any other rate.

Now note that in all the previous cases link $u$ was chosen to be any arbitrary link in $P^j_i$. Therefore, we can conclude that at every link in $P^j_i$ it is possible to increase $r^j_i$ by a non-zero positive amount without decreasing the rate $r^j_{i'}$ of some $V^C_i$ for which $r^j_{i'} \leq r^j_i$ while maintaining feasibility at the link. But this contradicts the maxmin definition.

(if part) Assume every leaf is either BLC or MRC. Consider the two cases: (1) $V^C_i$ is BLC at a link $u$ or (2) $V^C_i$ is MRC at a link $u$. 

99
Assume case (1). Then, in order to increase $r_i^j$ without violating the capacity constraint, one must decrease some other $r_i^{k'}$ such that $VVC_i^{k'}$ crosses link $u$ and $r_i^{k'} > m_i^{*k'}$. Because $VVC_i^j$ is BLC at link $u$, we have that $r_i^j = R_u \geq r_i^{k'}$. Now let $VVC_i^{j'} \in DS(VVC_i^{k'})$ be the leaf that satisfies $r_i^{j'} = r_i^{k'}$. We conclude that we cannot increase $r_i^j$ while maintaining feasibility at link $u$ without decreasing a leaf’s rate $r_i^{j'}$, where $r_i^{j'} \leq r_i^j$.

Assume case (2) where $VVC_i^j$ is MRC at a link $u$. There are two sub-cases: (2.1) link $u$ is saturated or (2.2) link $u$ is not saturated. Suppose that link $u$ is saturated (2.1). If we increase the rate $r_i^j$ we must drop some other rate in order to satisfy the feasibility condition. We cannot drop any VVC whose rate is at its minimum. Thus, the only rate we can drop must be from any $VVC_i^{k'}$ crossing link $u$ such that $r_i^{k'} > m_i^{*k'}$. But by the MRC and the advertised rate definitions we have

$$r_i^j \geq R_u = \max \{ r_i^{j'} | VVC_i^{j'} \in V_u, r_i^{j'} > m_i^{*k'} \}.$$  

This means that $r_i^{k'} \leq r_i^j$. Now let $VVC_i^{j'} \in DS(VVC_i^{k'})$ be the leaf that satisfies $r_i^{j'} = r_i^{k'}$. We conclude that we cannot increase $r_i^j$ while maintaining feasibility at link $u$ without decreasing a leaf’s rate $r_i^{j'}$, where $r_i^{j'} \leq r_i^j$. Finally, assume case (2.2) where link $u$ is not saturated. Then from Definition 5.12 we have that $R_u = \infty$. From the MRC assumption, we have $r_i^j = m_i^{*k} \geq R_u$, where $VVC_i^k$ crosses link $u$. Now note that this contradicts feasibility condition.
Theorem 5.1 gives us the first automatable optimality condition. This condition can be coded into a proper switch algorithm and a corresponding source algorithm that will let us later derive a distributed protocol. Now the next theorem, which introduces our second optimality condition, will provide us a mathematical formula for the source algorithm.

**Theorem 5.2. Multicast Projection Optimality Condition.** A leaf rate vector \( r \) is maxmin if and only if for every leaf \( VVC_i^j \) its rate satisfies the following condition,

\[
r_i^j = \max\{\min\{R_u^i | u' \in P_i^j\}, \max\{m_i^{*k} | VVC_i^{k} \in P_i^j\}\} \tag{5.7}
\]

We call the above expression the *multicast projection optimality condition*.

**Proof:** (only if part) Assume \( r \) is the maxmin rate vector and let \( VVC_i^j \) be an arbitrary VVC. Then there are two cases to consider: (1) \( VVC_i^j \) is MRC at a link \( u \) or (2) \( VVC_i^j \) is BLC at a link \( u \). Note that these are non-exclusive cases in the sense that the VVC can be MRC at a link and at the same time it can be BLC at some other link. Assume case (1). Then, we have \( r_i^j = m_i^{*k} = \max\{m_i^{*k} | VVC_i^{k} \in P_i^j\} \), otherwise condition 2 in feasibility definition (Definition 5.8) would not hold. Also, since \( r_i^j \geq R_u \) then we conclude that (5.7) is true. Consider case (2). There are two more cases: (2.1) \( R_u = \min\{R_u^i | u' \in P_i^j\} \), (2.2) \( R_u \neq \min\{R_u^i | u' \in P_i^j\} \). Assume case (2.1), then (5.7) is true since condition 2 in the feasibility definition implies that \( r_i^j \geq \max\{m_i^{*k} | VVC_i^{k} \in P_i^j\} \) and BLC condition implies \( r_i^j = R_u \). Assume now case (2.2), then exits a link \( u' \) such that \( R_u < R_u \) and
There are two more sub-cases: (2.2.1) \( r_i^{k'} = m_i^{*k'} \) or (2.2.2) \( r_i^{k'} > m_i^{*k'} \), where

\[ VVC_i^{k'} \in V_u. \]

If \( r_i^{k'} = m_i^{*k'} \) (2.2.1), then because of BLC condition we have \( r_i^j = R_u > R_{u^i}. \)

We also have \( F_u = C_u \), otherwise \( R_u = \infty \) and feasibility condition would not hold.

Then, we conclude that \( VVC_i^j \) is MRC at link \( u \) and this case is reduced to that of case (1).

Assume case (2.2.2), where \( r_i^{k'} > m_i^{*k'} \). Then from the advertised rate definition, we have \( R_u \geq r_i^{k'} \).

Since \( r_i^{k'} = r_i^j = R_u \), we must conclude \( R_u \geq R_{u^i} \), which is a contradiction.

(fif part) Assume that (5.7) is true and let \( u \) be a link such that \( R_u = \min \{ R_{u^i} \mid u^i \in P_i^j \} \).

There are two cases:

1. \( R_u > \max \{ m_i^{*k'} \mid VVC_i^{k'} \in P_i^j \} \) and

2. \( R_u \leq \max \{ m_i^{*k'} \mid VVC_i^{k'} \in P_i^j \} \).

Assume case (1). Then from (5.7) we have that \( r_i^j = R_u \) and \( r_i^j > m_i^{*k'} \), where

\[ VVC_i^{k'} \in V_u. \]

We also have that \( F_u = C_u \), otherwise \( R_u = \infty \) and feasibility condition would not hold. Now this means that \( VVC_i^j \) is BLC at link \( u \). Assume case (2), then from (5.7) we have that \( r_i^j \geq R_u \). There are two more cases: (2.1) \( r_i^j = m_i^{*k} \) or (2.2) \( r_i^j > m_i^{*k} \), with \( VVC_i^{k'} \in V_u \). If case (2.1) holds, then \( VVC_i^j \) is MRC at link \( u \). On the other hand, if case (2.2) holds, then from the advertised rate definition we have \( R_u \geq r_i^j \). Since we also had that \( r_i^j \geq R_u \), we conclude that \( r_i^j = R_u \) and, hence, \( VVC_i^j \) is BLC at link \( u \).
3 Multicast CPG Algorithm

In order to define an algorithm to compute the maxmin solution, we need to introduce the concepts of fair share and remaining fair share in a multicast sense.

**Definition 5.19. Fair Share.** The ratio $\frac{C_u}{N_u}$, where $N_u$ is the number of VVCs crossing link $u$, is called the *fair share* at link $u$. Intuitively, it is the fair share for all the VVCs crossing link $u$ while assuming none of them is constrained.

**Definition 5.20. Remaining Fair Share.** The ratio $(C_u - Fc_u)/(N_u - Nc_u)$ is called the *remaining fair share* at link $u$, where $Fc_u$ is the sum of rates for the constrained VVCs at link $u$ and $Nc_u$ is the number of constrained VVCs at link $u$, if $N_u - Nc_u > 0$.

### 3.1 Single-link Algorithm

Figure 3 presents an algorithm to solve the multicast maxmin problem for the single link case. In this algorithm, variables that are subject to change in each iteration are super-indexed with $l$, the iteration number.

**SolveSingleLinkCMMM (N)**

**Input parameters:**

$N$: a network with a single link $j$

**Output parameters:**

$\{r^j_i\}$: set of maxmin rates for the VVCs crossing link $j$

$R_j$: advertised rate for link $j$

1. set $l = 1$; Mark all VVCs as unconstrained;

2. For any unconstrained $VVC^j_i$ set $r^{j, i}_i = (C^j_i - Fc^j_i)/(N^j_i - Nc^j_i)$;
3. Calculate $R^i_j$. If $r^{i,j}_l = \max\{R^i_j, m_l\}$ for each VVC $j$ then set $R^l_j = R^i_j$, set $r^{l,j}_i = r^{i,j}_l$ for all $r^i_j$ and stop; Otherwise, set $r^{i,j}_l = \max\{R^i_j, m_l\}$ for any VVC $j$ such that $r^{i,j}_l \neq \max\{R^i_j, m_l\}$, mark these VVCs as constrained, do $l = l + 1$ and go to step 2;

Figure 3 Single link projection algorithm

Proposition 5.1. Convergence of the Single-link Projection Algorithm. The single-link projection algorithm presented in Figure 3 converges to the constrained maxmin solution in a finite number of steps.

Proof. We note first that a multicast network with a single link is as well a unicast network where the VVCs are changed by VCs. Then, since the $\text{SolveSingleLinkCMMM}$ algorithm is the same as the $\text{SolveSingleLinkCMM}$ algorithm, by Proposition 3.1 the above proposition must hold.

Let us examine now the cost of the single link algorithm.

Property 5.2. Single-Link Algorithm Cost. The worst-case cost of the Single-Link Projection Algorithm is $O(N_u)$, with $N_u$ the number of VVCs in link $u$.

Proof. From the algorithm, it must be true that at each iteration only one VVC becomes MRC. Therefore, since the algorithm converges to the maxmin solution, its cost must be at most $N_u$ iterations.
3.2 Multi-link Algorithm

Figure 4 presents an algorithm to compute the multicast maxmin solution of a multi-link network. The algorithm builds on the CPG algorithm presented in Chapter III, adding some new logics to support the more generic case of multicast flows.

$$\text{SolveNetworkCMMM (N)}$$

**Input parameters:**
- $N$: a network

**Output parameters:**
- $\{r_i^l\}$: set of maxmin rates at the VVC leafs
- $\{R_u\}$: set of advertised rates

0. $L = 1$;

1. For each link $u$ in $N$, obtain $R_u$ and $r_i^l$, for each VVC $i \in V_u$, by executing SolveSingleLinkCMMM(Link $u$);

2. For each link $u$ such that $R_u = \min\{R_u \mid \text{link } u \text{ and } u' \text{ share a VC}\}$ do the following:
   2.1. Add link $u$ to the set of level $L$ links;
   2.2. For each VVC $i \in V_u$:
      2.2.1. If $r_i^l > m_i^{\ast/l}$, then remove from the network VVC $i^\prime$ and any VVC belonging to $DS(VVC_i)$;
      2.2.2. If $r_i^l = m_i^{\ast/l}$, let $\Omega = \{\text{VVC}^k \mid \text{VVC}^k \in DS(VVC_i^l) \text{ and } m_i^{\ast/k} = m_i^{\ast/l}\}$. Then, remove from the network VVC $i^\prime$ and any VVC in $\Omega$. Now note that the set $DS(VVC_i^l) - \Omega$ defines a set of graphs and that each one of them will be treated in the next iteration as a new VC. Then, add an artificial link with capacity $m_i^{\ast/l}$ and extend the VVC source of the new VCs to cross this link;
   2.2.3. Assign $r_i^l$ to any VVC removed in steps 2.2.1 or 2.2.2;

2.3. For each link still in the network, reduce its capacity by the bandwidth assigned to the VVCs crossing it and removed in step 2.2;

3. If the set of VVC leaves in $N$ is not empty, do $L = L + 1$ and go to 1;

*Figure 4 SolveNetworkCMMM algorithm*
Theorem 5.3. Convergence of the CPG Algorithm. The rates computed by the SolveNetworkCMMM algorithm converge to the maxmin solution in a finite number of steps.

Proof. We need to prove two conditions: (1) Finiteness, the algorithm finishes in a finite number of steps; (2) Correctness, assuming the algorithm terminates then the output rates are maxmin.

To prove finiteness it is sufficient to show the following two conditions: (1.1) at the beginning of each iteration there exists at least one set of feasible rates and (1.2) at each iteration at least one link is removed. The first condition is needed so that the algorithm can be properly executed at each iteration while the second condition states that at each iteration the size of the problem is reduced by one unit or more. Therefore, after a finite number of iterations have been executed, if the two conditions hold the algorithm must terminate.

To prove condition (1.1) we need to understand how links at step 2 are chosen. The feasibility properties of the network at each iteration will be preserved if, at each iteration, it is possible to find a rate vector that satisfies conditions (1-2-3) of Definition 5.8. Note first that at a particular level \( L \), we only chose a link \( u \) to be included in such level if it has the minimal advertised rate among the links that share a VC with link \( u \) (step 2). Now let \( VVC^j_i \) be a VVC that crosses a link \( u \) that belongs to level \( L \) and consider \( VVC^k_i \) a VVC that is removed at iteration \( L \). Then from steps 2.2.1 and 2.2.2 there are two cases: (1.1.1) \( VVC^k_i \) is assigned a rate \( r^j_i = R_u > m^*_{ij} \) or (1.1.2) \( VVC^k_i \) is
assigned a rate $r_i^j = m_i^{*j} \geq R_u$. In what follows, we will prove that conditions (1-2-3) of Definition 5.8 are preserved at each iteration of the algorithm. In the first case (1.1.1) and recalling the advertised rate definition (Definition 5.12), a VVC that does not exceed the capacity requirements in the link with minimal advertised rate along its VC path will not exceed the capacity requirements of any other link that it crosses. Therefore, condition 1 is guaranteed. For the same case (1.1.1), conditions 2 and 3 are preserved since $r_i^j > m_i^{*j}$.

Assume now case (1.1.2). Since a network must always guarantee that it is possible to assign the minimal rate requirement to a VVC, the assignment $r_i^j = m_i^{*j}$ must preserve condition 1. In addition, this assignment trivially preserves condition 3. Finally, the inclusion of an artificial link in the network will guarantee the preservation of condition 2.

Condition (1.2) is trivial if we notice that the set of links defined at step 2 is always non-empty.

To prove correctness, consider an arbitrary leave VVC $VVC_i^j$ and its rate assigned by the CPG algorithm $r_i^j$. Let also $u$ be the link that caused $VVC_i^j$ to be removed and let $VVC_i^k$ be such that $VVC_i^k \in V_u$. Clearly, $P_i^j$ crosses link $u$ and therefore $r_i^j = r_i^k$. In addition, $F_u = C_u$ since the output rates from the single-link algorithm always saturate the input link. We know that $r_i^k$ can only take two values: (1) $r_i^k = R_u > m_i^{*k}$ or (2) $r_i^k = m_i^{*k} \geq R_u$. But from definitions 5.14 and 5.15, cases (1) and (2) imply that the leaf is BLC and MRC, respectively. Now applying theorem 5.1 we must conclude that the output rates from the CPG algorithm are maxmin.
4 Examples

Consider the network configuration in Figure 5. In this figure, VCs and VVCs are labeled with their name followed by the minimal rate constrain of their associated destination, if any. Links are also labeled with their available capacity. Let us use our multicast theory to make a maxmin analysis of this network. We will first compute the maxmin rates by using the CPG algorithm. Then, correctness of the rate assignment will be checked by using the bottleneck and the projection optimality conditions.

Table 1 shows the results of executing the CPG algorithm to our network sample. Each row represents one iteration. Values for the advertised rate at each link and the actions to perform are presented for each iteration. Cells in a row \( i \) that are shadowed represent links that are removed at iteration \( i \). Though the number of bottleneck links is equal to 5, note that it only takes three iterations for the algorithm to converge. This is because step 3 in the CPG algorithm can be executed in parallel for more than one link.
<table>
<thead>
<tr>
<th></th>
<th>AR(L1)</th>
<th>AR(L2)</th>
<th>AR(L3)</th>
<th>AR(L4)</th>
<th>AR(L5)</th>
<th>AR(L6)</th>
<th>AR(L7)</th>
<th>AR(L8)</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>It 1</td>
<td>95</td>
<td>15</td>
<td>150</td>
<td>36.6</td>
<td>60</td>
<td>150</td>
<td>5</td>
<td>10</td>
<td>Remove L7, L8, VVC(^5), VC(_2), VC(_3); set (r(_1) = 115), (r(_2) = 10) and (C(_1) = 200), (C(_2) = 150), (C(_3) = 100).</td>
</tr>
<tr>
<td>It 2</td>
<td>200</td>
<td>35</td>
<td>150</td>
<td>50</td>
<td>60</td>
<td></td>
<td></td>
<td></td>
<td>Remove L2, L3, VVC(_1), (VVC(_1) ), VC(_2); set (r(_3) = 150), (r(_1) = 115), (r(_2) = 35) and (C(_2) = 75).</td>
</tr>
<tr>
<td>It 3</td>
<td>200</td>
<td>100</td>
<td></td>
<td>75</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Remove L5, (VC(_3)), (VVC(_5); set (r(_4) = 60), (r(_4) = 60).</td>
</tr>
</tbody>
</table>

Table 1 CPG algorithm execution

Note that from Table 1 links 1 and 4 do not have a shadowed cell since they are never removed from the network when executing the CPG algorithm. Links that are never removed correspond to pseudo-saturated links (Definition 5.18). For these links, the crossing VVCs are all constrained somewhere else and their capacity is not fully utilized.

In Theorem 5.1 we proved that a multicast rate assignment is maxmin if and only if any leaf in the network is either BLC or MRC. Table 2 gives the results when checking the
optimality condition for each VVC. Note that any VVC leaf is either BLC or MRC at least at one link, which confirms the optimality of the solution.

<table>
<thead>
<tr>
<th>VVC$^1_1$</th>
<th>VVC$^2_1$</th>
<th>VVC$^3_1$</th>
<th>VVC$^4_1$</th>
<th>VVC$^5_1$</th>
<th>VC$^2_2$</th>
<th>VC$^3_3$</th>
<th>VC$^4_4$</th>
<th>VC$^5_5$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rate</td>
<td>150</td>
<td>115</td>
<td>150</td>
<td>60</td>
<td>115</td>
<td>35</td>
<td>60</td>
<td>5</td>
</tr>
<tr>
<td>Constrained</td>
<td>not a leaf</td>
<td>not a leaf</td>
<td>BLC at L3</td>
<td>BLC at L3</td>
<td>MRC at L7 and L2</td>
<td>BLC at L2</td>
<td>BLC at L5</td>
<td>BLC at L7</td>
</tr>
</tbody>
</table>

*Table 2 Bottleneck optimality condition check*

It can be seen that the projection optimality condition presented in theorem 2 also holds. We leave the checking of this condition to the reader.

Let us now consider the network in Figure 6, consisting of two multicast connections, VC$^1_1$ and VC$^2_2$. In this example, we will show the concept of *artificial link*. Consider that VC$^1_1$ has two receivers D$^{1,1}_{1,1}$ and D$^{1,2}_{1,2}$ and VC$^2_2$ has two more receivers D$^{2,1}_{2,1}$ and D$^{2,2}_{2,2}$. All the receivers have a minimal rate constraint of zero except for D$^{2,2}_{1,2}$, which requires at least a rate of 40. The network consists of 5 links labeled L$i$, with $1 \leq i \leq 5$, and each of the links has a capacity showed by the number in brackets.
Figure 7 shows how the CPG algorithm iterates. It requires three iterations. At the first iteration, L1 becomes a bottleneck with an advertised rate of 10. Following step 2.2.1 we have that D_{1,2} is assigned a rate of 40 while following step 2.2.2 we have that D_{2,1} is assigned a rate of 10 and an artificial link with capacity 40 (the minimal rate constraint of D_{1,2}) is added to VC1 and labeled AL6. At the second iteration, L5 becomes a bottleneck with advertised rate 15 and D_{2,2} is assigned a rate of 15. Finally, at iteration 3 L2 becomes a bottleneck with a rate of 25 and D_{2,1} is assigned a rate of 25.
Figure 7 CPG iterations to solve the network in Figure 6
5 Distributed Protocol

In this section, we present a distributed protocol that achieves maxmin rates in multicast networks. First, we present the source, destination and switch algorithms. Then, we prove the convergence of the protocol. Simulations at the end of this section are provided to empirically show the behavior of the protocol.

5.1 Source, Destination and Switch Algorithms

In our multicast signaling protocol, at each branching point of a multicast tree, forward resource management (RM) packets that bring feedback information are duplicated and sent to all possible downstream VVCs. Upon arriving at a destination, RM packets are turn around back to the source becoming backward resource management packets. In order to avoid implosion of BRM packets at the source node, backward RM packets have to be coalesced at each branching point as they travel their way back to the source. A decision on when to consolidate the feedback has to be done. If this decision is done too early, feedback from all the branches may have not arrived yet, inducing a noisy feedback value. On the other hand, if the consolidation occurs too late, backward RM packets arrive at the source experiencing higher latencies and defeating the purpose of their trip. This is known as the consolidation noise problem.

Several authors have addressed this issue (for a good review, refer to [FAH98]). While the consolidation noise problem is out of the scope of this lexicographic theory, for the sake of protocol completeness we refer to [ROS00b] for a mathematical result that leads to an efficient solution of the problem. In the remaining part of this chapter, we shall
assume that consolidation of backward RM packets is achieved by the means of one of the protocols in the literature.

To design a distributed multicast protocol, three components have to be defined: the behavior of the source, the behavior of the switch and the behavior of the destination. Note that we will not differentiate between the behavior of branching switches and non-branching switches, since the latter can be treated as the special case of the former where the number of branches is one.

Figure 8 presents the source algorithm. Upon the reception of a backward RM packet, the source adapts its transmission rate (TR) to the value brought by the ER field in the packet. In addition, the source periodically sends forward RM packets. In this implementation, we assume sources are greedy and, therefore, the ER field is initialized to infinity. If that is not the case, then the ER field can be set to the peak rate value of the source.

sourcealgorithm()

When a RM packet arrives
TR ← RM.ER;
When is time to send a RM packet
RM.ER ← ∞ ;
Send RM packet downstream;

Figure 8 Source algorithm

Figure 9 presents the destination algorithm. Upon receiving a forward RM packet, the ER field is set to infinity and the packet is sent back to the source.
DestinationAlgorithm()
When a RM packet arrives
   RM,ER ←∞;
   Send RM packet upstream;

Figure 9 Destination algorithm

Before presenting the switch algorithm, Figure 10 introduces the flow architecture of the multicast switch. In this configuration, the multicast switch has four input ports (i-ports) and four output ports (o-ports). We assume that data flows from left to right. There are currently two multicast VCs flowing through the switch. Flow 1 arrives from i-port 1 and departs from o-ports 1, 3 and 4. Hence, this multicast VC contends for resources $C_1$, $C_5$, $C_7$ and $C_8$. Similarly, flow 2 arrives from i-port 4 and departs from o-ports 2 and 4, contending for resources $C_4$, $C_6$, and $C_8$. Notice also that in this example, the two flows will interact when contending for resources in o-port 4, that is $C_8$. Such is the type of contention that our maxmin protocol will resolve.
Figure 10 Flow architecture of the multicast switch

Figure 11 corresponds to the switch algorithm. When a connection is set up, the number of VCs that cross the switch \((N)\) is incremented by one and memory is allocated for several fields. The following is a description of these fields:

- **UB\(_i\):** Stores the last ER value received from the upstream port of the switch for VC \(i\).

- **DB\(_{ij}\):** Stores the last ER value received from the downstream port of the switch for branch \(j\) of VC \(i\).

- **W\(_{ij}\):** Stores the minimum of **DB\(_{ij}\)** and the last advertised rate computed for branch \(j\) of VC \(i\).

- **MFR\(_{ij}\):** Stores the minimum flow rate for branch \(j\) of VC \(i\).

When a connection is closed, the memory reserved for the connection is released and \(N\) is decremented by one.
**SwitchAlgorithm()**

When a connection flow $i$ is set up with $M_i$ branches in this switch

Allocate new entries for $UB_i$, $DB_i^j$, $W_i^j$, $MFR_i^j$ for $j = 1...M_i$;
Set $MFR_i^j$ to the minimal rate allowed by each branch $j$;

$N \leftarrow N + 1$;

When a connection is closed
Free memory space reserved for the connection;

$N \leftarrow N - 1$;

When a forward RM packet arrives from flow $i$

$UB_i \leftarrow RM.ER$;

For each downstream branch $j$
Create a copy of the packet;
$p \leftarrow$ port associated with branch $j$
$AR = ComputeAR(i, p)$;
$RM.ER \leftarrow \max\{\min\{AR, UB_i, DB_i^j\}, MFR_j\}$;
Forward RM packet to the downstream branch $j$;

When a backward RM packet arrives from flow $i$ and branch $j$

$p \leftarrow$ outgoing upstream port for this packet;
$DB_i^j \leftarrow RM.ER$;
$AR = ComputeAR(i, p)$;
$W_i^j = \min\{DB_i^j, AR\}$;
$W = \max\{W_i^j, j = 1...M_i\}$;
$RM.ER \leftarrow \max\{\min\{AR, W, UB_i\}, MFR_j\}$;
Forward RM packet to the upstream branch;

---

**Figure 11 Switch algorithm**

The actual signaling algorithm is executed every time an RM packet arrives. Upon the reception of a forward RM packet for VC $i$, its ER value is stored in $UB_i$ and after updating its ER field using the procedure $ComputeAR$, a copy of the packet is forwarded to each of the VVC branches. Upon the reception of a backward RM packet from $VVC_i^j$, its ER value is stored in $DB_i^j$. After computing a new value for the ER field, the packet is forwarded upstream. The $ComputeAR(i,p)$ procedure calculates the available rate at the outgoing port $p$ for VC $i$. 
Property 5.3. Bi-directional minimization, Transient oscillation freedom. Since the multicast protocol presented in this section is based on the unicast protocol developed in 0, it will inherit Property 4.1 and Property 4.2.

We have seen how the information that is distributed throughout the network is transported and stored in each of the network components (source, switch and destination). The following section describes the rate allocation algorithm, ComputeAR(i,p).

5.2 Rate Computation Algorithm

The objective of the rate computation algorithm ComputeAR(i,p) is to gather the network status received from the signaling protocol and to compute the AR value (available rate) for VC at port p. In this procedure, each VVC has a minimal flow rate \( MFR \). In addition, a switch should not assign a rate to a VVC that is larger than the rate that can be afforded by other links crossed by the same VVC. The information of the rate that can be afforded by other links is given by the ER field in the RM packets. We consolidate the feedback from both upstream (forward RM packets) and downstream (backward RM packets) into a single parameter \( B \) as follows:

\[
\text{If VVC is a branch at switch s, then}\]
\[
B_i \leftarrow \min\{UB_i, DB_i\};
\]

\[
\text{If VVC is a root at switch s, then}\]
\[
B_i \leftarrow \min\{UB_i, \max\{DB_i, j=1...M_i\}\};
\]
$B_{i,j}$ is to be interpreted as the maximum bandwidth that other switches different than switch $s$ can afford to assign to $VVC_{i,j}$. If $VVC_{i,j}$ is a branch, then it is assigned the minimum of the last ER value received from downstream and upstream. If $VVC_{i,j}$ is a root, then it is assigned the minimum of the last ER value that has been received from upstream and the maximum of the last ER values received from all the downstream branches. This reflects the fact that we are concerned with multi-rate trees and, therefore, the root branch should try to satisfy the branch that can assign the larger rate. In summary, $ComputeAR(i,p)$ has to resolve a single link maxmin rate problem with both minimal flow rate constraints given by $MFR_{i,j}$ and peak flow rate constraints given by $B_{i,j}$, as shown in Figure 12a. Because the super-index $j$ remains a constant throughout the discussion in this section, for the sake of simplicity we omit it.

Following the same approach that we took in the unicast protocol (0), we transform our problem into an equivalent one: a multi-link problem with no peak flow rate constraints (Figure 12). As shown in Figure 12b, any peak flow rate of a flow $i$ is removed by introducing a new link connected to a switch with capacity equal to the peak flow rate of flow $i$ and forcing flow $i$ to cross this link.
Now, our problem is that of solving a multi-link network where we know all of its parameters. Therefore, one can use the CPG centralized algorithm to solve it. Figure 13 presents an implementation of the ComputeAR. The reader can check that this implementation is the same as the CPG algorithm but for the particular case of our problem in Figure 12b.

ComputeAR()

Input parameters:
\( i \): VC for which we compute AR;
\( p \): Switch port where we compute AR;

Output parameters:
\( \text{AR} \): Advertised Rate;

Local parameters:
\( \Omega \): Set of VVC rates bottlenecked at their MFRs;
\( \Psi \): Set of VVC rates bottlenecked somewhere else;
\( C \): Link capacity for port \( p \);
\( N \): number of VCs using port \( p \);
\( \text{RC} \): Remaining capacity;
\( \text{RN} \): Number of remaining flows (those that are not in \( \Omega \cup \Psi \));
\( j \): is such that \( \text{VVC}_i \) is incident to port \( p \);
B_i: peak rate constraint enforced by the rest of the network;
M_i: number of branches for VC_i;

1. For each VVC_i crossing port p:
   - If VVC_i is a branch, B_{i,j} \leftarrow \min\{UB_i, DB_{i,j}\};
   - If VVC_i is a root, B_{i,j} \leftarrow \min\{UB_i, \max\{DB_i^k, k=1\ldots M_i\}\};

2. \(\Psi \leftarrow \emptyset;\) RC \leftarrow C; RN \leftarrow N;
3. \(\Omega \leftarrow \emptyset;\)
4. AR \leftarrow RC/RN;
5. If \(\exists i \in (\Omega \cup \Psi)\) such that \(AR < MFR_i\)
   - Put any flow i such that \(i \in (\Omega \cup \Psi)\) and \(AR < MFR_i\) into \(\Omega\);
   - RC \leftarrow C - \sum_{i \in \Omega} MFR_i - \sum_{i \in \Psi} B_{i,j}; RN \leftarrow N - |\Omega \cup \Psi|;
   - Return to 4;
6. If \(\exists i \in (\Omega \cup \Psi)\) such that \(AR > B_{i,j}\)
   - Put any flow i such that \(i \in (\Omega \cup \Psi)\) and \(AR > B_{i,j}\) into \(\Psi\);
   - RC \leftarrow C - \sum_{i \in \Psi} B_{i,j}; RN \leftarrow N - |\Psi|;
   - Return to 3;
7. Stop;

Figure 13 ComputeAR Procedure to solve the single-link case

Lemma 5.1. Multicast link convergence condition. Let \(AR_j^1\) be the advertised rate computed at the first iteration of the CPG centralized algorithm for an arbitrary link \(j\). Then, given any arbitrary values of the fields \(B_j,\) the rate computed by the procedure ComputeAR is always greater or equal than \(AR_j^1\).

Proof. Since the single-link problem for multicast networks is equivalent to the single-link problem for unicast networks, for a formal proof refer to Lemma 4.1.
5.3 Protocol Convergence

Let us now prove the convergence of the multicast distributed protocol.

*Theorem 5.4. Network Convergence.* Given arbitrary initial conditions on the state of the links and the state of the RM packets in transit, the proposed distributed algorithm converges to the maxmin rates as long as the set of sessions and the available bandwidth eventually stabilize.

*Proof.* Let $t_0$ be the time at which the set of sessions and the available bandwidth stabilize. From Lemma 5.1 we know that the AR value computed independently at an arbitrary link $j$ is greater or equal than the advertised rate computed at the first level of the centralized algorithm for link $j$, $AR^{1}_j$. Let $D$ be an upper bound on the one-way delay (half the round trip delay) in the whole network. Then the time required for a link to know about the change of the state of another link, both sharing a flow, is at most $D$. Let link $k$ be an arbitrary first level bottleneck. From the centralized algorithm, we have that $AR^{1}_j \geq AR^{1}_k$ for any link $j$ that shares a flow with link $k$. Then, at time $t_0 + D$ we have that the state of the link $k$ satisfies $B^{1}_i \geq AR^{1}_k$ for any $i$. Now when executing algorithm ComputeAR at link $k$ we have that the first time step 6 is executed, its checking condition will not hold and the algorithm will finish returning $AR = AR^{1}_k$. Then, at time $t_0 + D$ all first level bottleneck links have converged to their maxmin advertised rate. This means that at time $t_0 + 2D$ any link will be aware of the fact that the first level bottlenecks have converged. Moreover, let $i$ be a flow that is bottlenecked at the first CPG level. Since $AR^{1}_j \geq AR^{1}_k$ for any link $j$ that shares a flow with link $k$, then the value of $B_i$ will be set.
to its maxmin rate and will not be changed anymore. Therefore, any time after $t_0 + 2D$, when a link $j$ that is not a first level link computes AR, it will find that these first level flows are constrained somewhere else (in a first level bottleneck). Now we can remove these flows from the network and subtract their assigned rate to the links they cross. Applying Lemma 5.1, we know that the AR value computed independently at an arbitrary link $j$ is greater or equal than the advertised rate computed at the second level of the centralized algorithm for link $j$, $AR_j^2$, which means that we are in the same conditions as we were at the beginning of this proof. Hence, by induction Theorem 5.4 must hold.

§

5.4 Simulations

In this section, we present a simulated implementation of the multicast protocol. Figure 14 shows the network configuration. It consists of one multicast flow interacting with two unicast flows. In order to consider the case of dynamic bandwidth, a higher priority flow is also considered. The switches only serve low priority traffic whenever there are no high priority packets pending to be processed. Note that the multicast protocol only runs on the low priority flows, determining the bandwidth that needs to be allocated to each of these flows in a dynamic manner.
Each link is assigned a capacity C expressed in Mbps. In order to consider network delays, each link is also configured to have a length of 10 Kms. Finally, in order to evaluate the effects of the minimal rate constraint, we have added a MFR of 50 Mbps to destination D(1,1).

The high priority traffic is configured as an ON-OFF traffic, with a rate of 60 Mbps during the ON interval and a rate of 0 Mbps during the off interval. Each interval has a length of 5 milliseconds.

The following table shows the maxmin rate solution for each flow and for each interval.

<table>
<thead>
<tr>
<th>rate</th>
<th>S(4)</th>
<th>S(1)</th>
<th>D(1,1)</th>
<th>D(1,2)</th>
<th>S(2)</th>
<th>S(3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Off</td>
<td>0</td>
<td>70</td>
<td>70</td>
<td>60</td>
<td>80</td>
<td>70</td>
</tr>
<tr>
<td>On</td>
<td>60</td>
<td>60</td>
<td>50</td>
<td>60</td>
<td>90</td>
<td>30</td>
</tr>
</tbody>
</table>

*Table 3 Maxmin rate solution*

Figure 15 shows the transmission rates for each flow obtained when running the simulator for a period of 25 milliseconds. The reader can see that after transient states, the rates converge to the theoretic maxmin solution.
Figure 15 Transmission rates for each of the low priority flows

Figure 16 shows the size of the queue at switch sw3 for traffic forwarded to switch sw1. This is the most critical queue since it is directly affected by the high priority flow. The graph shows that the queue grows up to 2.5 KB of data during a transition from off to on. The small queue size is because the transient states induced by the protocol are very short.
Figure 16 Queue size at switch sw3 for traffic forwarded to sw1
Chapter VI

MAXMIN LAMBDA ALLOCATION FOR DWDM NETWORKS

“Share – tools, ideas. Trust your colleagues”

Rules of the Garage, 4th rule

1 Introduction

Advances in optical component technology have made the deployment of wavelength routing possible in today’s networks. However, as electronic network is slow but intelligent, and the optical network is fast but dumb, a hybrid approach of combining electronic switching/forwarding and light-path switching is preferred [BAN00]. When this approach is adopted, the problem is to detect flows (mostly coarse-grain) of sufficient intensity and duration to merit light-path switching, and to assign wavelengths to the flows in an appropriate way. This chapter proposes the use of discrete maxmin rate allocation to determine the assignment.
The discrete maxmin problem was first formulated by Sarkar and Tassiulas [SAS00], while this author independently formulated the identical optimality condition\textsuperscript{6}. Sarkar and Tassiulas used the discrete maxmin condition to solve the multi-rate multicast allocation problem [SAS00] and we formulated the condition to solve the wavelength assignment problem for optical sub-networks. While the two results are complementary to each other, in this chapter, we show that the discrete maxmin optimality condition can be used as a unifying maxmin condition for both the discrete and continuous cases. In addition, the theory that follows includes a novel heuristic for solving the discrete lexicographic optimal rates for DWDM-based (Dense Wavelength Division Multiplexing) optical sub-networks.

Among the many discrete maxmin solutions for each assignment problem, lexicographic optimal solutions can be argued to be the best in the sense of "true maxmin." However, the problem of finding discrete lexicographic optimal solutions is known to be NP-complete [SAS00]. The heuristic proposed in this chapter, called LEX, is tested against all possible networks such that $|\Gamma|+|\Omega| \leq 10$, where $\Gamma$ and $\Omega$ are the set of links and the set of flows of the network, respectively. Out of 1,084,112 possible networks, the heuristic produces the exact lexicographic solutions except for 2,069 cases. Thus, the heuristic produces the true optimal solution for the tested networks with a success rate of 99.8%. Furthermore, for 0.2% cases where the solutions are non-optimal,

\textsuperscript{6} Our definition of discrete fairness is the same as that of maximal fairness in [SAS00] and our bottleneck optimality condition corresponds to their pseudo-bottleneck lemma for the case of DWDM networks
99.8% of these solutions are within the minimal possible distance from the true lexicographic optimal solutions (one single wavelength away from the optimal solution). The LEX heuristic is presented under the framework of the \textit{d-CPG} algorithm. This algorithm is a natural extension of the theory that we have derived in the previous chapters.

This chapter is organized as follows. Section 2 introduces optical networks and motivates the use of discrete maxmin to assign wavelengths. Section 3 presents a general theory of discrete maxmin and discrete lexicographic optimization. The \textit{d-CPG} algorithm is described in Section 4. The heuristic \textit{LEX} is described in Section 5 and the numerical experimentations are presented in Section 6.

2 \textit{Optical Networks Review and the Discrete Maxmin Problem}

Optical networks and electronic networks present different characteristics. Current interests on optical networks are based on the fact that, by using optical links, the amount of available bandwidth in the network can dramatically increase. However, optical networks also impose additional properties that are not as desirable, the main one being that current state-of-the-art for nonlinear optical operations is not as advanced as we would need. Some of these operations like reading and writing packet headers are critical for the design of packet switching networks. On the other hand, electronic processing is ideal for complex nonlinear operations. These operations allow for the design of many different protocols and this has provisioned the success of packet switched networks for the last 30 years. The downside of the electronic approach is its limited speed of transmission causing the so-called "electronic bottleneck."
Because the two previous conditions are expected to persist at least into the near future, current efforts on optical networks design have been focused on how to couple both technologies, optical and electronic, into a single network. An example of this effort is the Multiprotocol Lambda Switching [GMP03] approach proposed by the IETF.

In this section, we will review some of the issues concerning the optical-electronic model. We will first present the switch architecture, a key element to understand the capabilities of the network. Then, we will introduce the maxmin wavelength assignment problem.

2.1 Optical Switch Architecture

Figure 1 presents a functional diagram of the optical switch architecture. When the switch is initialized, all the incoming wavelengths are switched to the electronic layer through an optical-to-electronic converter. In the electronic layer, classic IP routing is carried out. Then, an electronic-to-optical converter brings data back to the optical format and the resulting wavelengths are sent out.

Now assume that the flow analyzer detects the existence of a large flow of data. Suppose that this flow comes in from a fiber A and goes out through a fiber B. Then the flow analyzer algorithm will try to find one or more unused wavelengths on both incoming fiber A and outgoing fiber B. Assuming wavelengths 1 and 2 on both fibers are available, respectively, then the optical switch is signaled to directly connect these wavelengths. That is, any data coming on lambda 1 from fiber A will be switched to lambda 2 in fiber B, creating a pure optical link. Finally, once the flow rate decreases
down to a certain threshold, the flow analyzer may decide to release the connection and return to the original mode with electronic switching.

![Flow analyzer Routing protocol](image)

**Figure 1 Optical switch architecture**

2.2 *The Maxmin Wavelength Assignment Problem*

We have seen that an algorithm to identify an available wavelength needs to be implemented inside the switch. In fact, the network is an interconnection of switches with the same or similar capabilities. Setting only one switch to use its optical layer switching is useless if the rest of switches for which the flow goes through are using electronic layer switching, since the later becomes the bottleneck. Therefore, before an optical switch can decide to set up a pure optical path through its fabric, it has to convince the other optical switches involved in the same flow to use the pure optical path. During this process, the switches have to agree on the set of wavelengths used. For that, a distributed protocol is required to solve the wavelength assignment problem.
Let us now consider the following problem. The left most network in Figure 2 shows a situation where switch 1 ($s_1$) has an optical path switched for flow A. The four available wavelengths in the switch (here denoted as $W = 4$) are assigned to this flow. For switches 2 ($s_2$) and 3 ($s_3$) no optical path is set and electronic forwarding is required for any packet going through. Now assume flow B is detected (the central graph in Figure 2). Then, three wavelengths are assigned to this new flow. Note that we cannot allocate more than three since this is the number of available wavelengths at switch 2. Finally, assume that flow C is now identified. In order to avoid starvation, switches 1 and 2 should reallocate the wavelengths assigned to each of their flows. Now the decision for switch 2 is whether to assign two wavelengths to flow C and 1 wavelength to flow B or vice-versa. Note that if two wavelengths are assigned to flow C, then the number of wavelengths allocated to flow A has to be reduced to two. Instead, if one wavelength is assigned to flow C, then three wavelengths can be provided to flow A. In conclusion, as shown in Figure 2, we have two possible rate allocations: $r_1 = [2,1,2]$ and $r_2 = [3,2,1]$. When all the other factors are the same, we should use $r_2$ since it better utilizes the resources in the network.
The above problem can be seen as a maxmin optimization problem. The main difference between the optical case and classical maxmin is that the former restricts the set of rates allocated in a flow to be an integer value. Because of this new constraint, the maxmin problem becomes harder to solve. In fact, [SAS00] proves that this is an NP-complete problem.

In what follows, we will derive a theory of discrete maxmin optimization and present a heuristic that allows for a feasible implementation of a maxmin protocol in optical networks. Numerical experimentations are presented at the end that show the merits of this heuristic.

3 Discrete Maxmin Theory

Let us first recall the classic maxmin fairness problem.

**Definition 6.1. Feasibility Condition.** Consider a network \((\Gamma, \Omega)\), where \(\Gamma\) is the set of its links and \(\Omega\) is the set of flows that we want to set as fixed paths in the network. Let \(W_j\) be the number of available wavelengths at link \(j\) and let \(r\) be a rate allocation vector. We say that \(r\) is a feasible rate vector if \(F_j = \sum_{i \in V_j} r_i \leq W_j\) for any link \(j\), where \(F_j\) is the total aggregated flow rate at link \(j\) and \(V_j\) is the set of flows crossing that link.
**Definition 6.2. Classic maxmin fairness.** We say that $r$ is a maxmin rate solution if and only if for each flow $i$ we cannot increase $r_i$ while maintaining feasibility without decreasing $r_j$ for some other flow $j$ for which $r_i \leq r_j$. More formally,

$$\left[ r, r' \in R_+^{|\Omega|} \text{ are feasible,} \quad r \text{ is max min } \right] \iff r', r_i > r_i' \Rightarrow r_j' < r_j \text{ and } r_j \leq r_i, \text{ for some } j \quad (6.1)$$

where $R_+^{|\Omega|}$ is the $|\Omega|$-dimensional field of positive real numbers.

In the discrete case, we have the constraint $r \in \mathbb{Z}_+^{|\Omega|}$, where $\mathbb{Z}_+^{|\Omega|}$ is the $|\Omega|$-dimensional ring of positive integer numbers. Let us now see why condition (6.1) cannot be applied to the discrete case. For that, consider Figure 3 where there are three flows crossing the same link. Assume the link has 4 available wavelengths. To reach the optimal solution in terms of both bandwidth utilization and fairness, let’s first allocate 1 wavelength to each of the flows. This gives us a fair allocation but not optimal in terms of resource utilization since one of the wavelengths is not used. The decision of where to assign the extra wavelength could be done according to some pricing scheme or some prioritization decision. In this theory, we will assume that all flows have the same right to get the remaining set of wavelengths, which means that one can pick an arbitrary flow and assign to it the extra wavelength. Therefore, any of the assignments $[1, 1, 2], [1, 2, 1]$ and $[2, 1, 1]$ are discrete maxmin solutions.
Let us now check definition 6.2 on one of our solutions, let’s say [2, 1, 1]. Consider an increase of bandwidth for flow 2. Since we are in the discrete case, the minimal increase is 1. To achieve this, we can decrease the rate of flow 1 by one unit so that we get the new allocation [1, 2, 1]. This means that we can increase the bandwidth of one flow without necessarily decreasing another flow rate that is lower or equal. Therefore, our set of discrete solutions does not satisfy the classic maxmin definition.

From the previous example, we conclude that a new definition of maxmin fair allocation is required when considering the discrete case.

**Definition 6.3. Discrete Maxmin Fairness.** We say that \( r \) is a **discrete maxmin rate** solution if and only if for each flow \( i \) we cannot increase \( r_i \) while maintaining feasibility without decreasing \( r_j \) to a value lower than the new value of \( r_i \), for some other flow \( j \).

More formally,

\[
\begin{bmatrix}
\text{r, } r' \in Z_+^{|\mathcal{Q}|} \text{ are feasible,} \\
\text{r is discrete max min}
\end{bmatrix} \iff r'_i > r_i \Rightarrow r'_j < r_j \text{ and } r'_j < r'_i, \text{ for some } j
\]

(6.2)

One can now check that any solution of the network in Figure 3 satisfies the discrete maxmin definition.

Note that in order to let solution [2, 1, 1] classify as discrete maxmin, we had to weaken the maxmin definition. Indeed, looking at expressions (6.1) and (6.2), the condition \( r'_j < r'_i \) is weaker than \( r_j \leq r_i \), under the assumption that \( r'_j < r_j \) and \( r'_i > r_i \). However,
the following theorem proves that, when applied to the continuous domain, the discrete definition is equivalent (and hence as strong) as the classic maxmin definition.

**Lemma 6.1. Unified maxmin definition.** The discrete maxmin definition applied to the continuous domain ($R_{+}$) is equivalent to the classic maxmin definition. That is, $r$ is a maxmin rate solution in the continuous domain (continuous maxmin) if and only if for each flow $i$ we cannot increase $r_i$ while maintaining feasibility without decreasing $r_j$ to a value lower than the new value of $r_i$, for some other flow $j$.

**Proof.** (Only if): Assume that $r$ satisfies the classic maxmin definition. Then, for any other feasible rate vector $r'$ we have that $r_i' > r_i \Rightarrow r_j' < r_j$ and $r_j \leq r_i$, for some $j$. The later means that $r_j' < r_i'$ which means that $r$ satisfies the discrete maxmin condition.

(If): Assume that $r$ satisfies the discrete maxmin condition in the continuous domain $R_{+}$. Let $i$ be an arbitrary flow. If we increase its rate to $r_i'$, i.e. $r_i' > r_i$, then there must exist a set of flows $j_1, j_2, \ldots, j_n$ that need to decrease their rates to $r_{j_1}', r_{j_2}', \ldots, r_{j_n}'$, respectively, and such that $r_i < r_i', r_{j_1} < r_{j_1}', r_{j_2} < r_{j_2}', \ldots, r_{j_n} < r_{j_n}'$. Let us define $\varepsilon$ as $\varepsilon = r_i' - r_i$. In addition, let us arbitrarily choose one of these flows, let’s say $j_k$, and let us assume that $r_{j_k} > r_i$. In order to maintain feasibility, it suffices that $r_{j_k} - r_i' = \varepsilon$. Now since the rates belong to the continuous space of the non-negative real numbers, we can always find a value of $\varepsilon$ small enough such that $r_{j_k}' = r_{j_k} + \varepsilon > r_i'$. But this contradicts the assumption that $r$ is discrete maxmin. Therefore, we reach a contradiction and it must be that $r_{j_k} \leq r_i$, which implies that $r$ satisfies the classic maxmin definition.
Lemma 6.1 shows that the discrete definition provided in this paper unifies both continuous and discrete domains. Classical literature has always referred to the maxmin solution as the rate vector that satisfies definition 6.2. However, we have seen that such definition is not applicable to the discrete case. On the other hand, Lemma 6.1 shows that the discrete maxmin definition is applicable to both discrete and continuous domains. This approach simplifies the maxmin framework in the sense that we do not need a different definition for each domain.

**Definition 6.4. Discrete Bottleneck Condition.** A flow $i$ is said to be bottlenecked at link $u$ if,

1. Flow $i$ crosses link $u$,

2. Link $u$ is fully utilized, i.e. $F_u = \sum_{i \in V_u} r_i = W_u$ and

3. $r_i \geq \max\{r_j, \forall j \in V_u\} - 1$

**Theorem 6.1. Bottleneck Optimality Condition.** A rate vector $r$ is discrete maxmin if and only if every flow is bottlenecked at some link.

**Proof.** (If): Suppose that every flow is bottlenecked at some link. Then, for any flow $i$ let link $u$ be its bottleneck. From the discrete bottleneck definition, we have that $r_i \geq \max\{r_j, \forall j \in V_u\} - 1$. Assume we increase $r_i$ by one unit, that is $r'_i = r_i + 1$. Then,
since the link is fully utilized, we have to decrease some other link rate by one unit, that is \( r_j' = r_j - 1 \). Thus, we have that \( r_j' < \max\{r_j, \forall j \in V_u\} \leq r_i' \), which proves the condition.

(Only if): Suppose that \( r \) is a discrete maxmin fair solution. To arrive at a contradiction, assume that there exists a flow \( i \) that has no bottleneck link. Consider every link \( u \) along its path. Then, we have two cases: (1) link \( u \) is fully utilized and (2) link \( u \) is underutilized. Let’s consider case (1) and let \( j \) be a flow such that \( r_j = \max\{r_j, \forall j \in V_u\} \).

Since \( u \) is not a bottleneck for flow \( i \), we have that \( r_i < r_j - 1 \). This means that at this particular link, we can increase \( r_i \) by one unit, \( r_i' = r_i + 1 \), and decrease \( r_j' = r_j - 1 \) and still \( r_i' \leq r_j' \). Assuming case (2), we have that we can increase \( r_i \) by one unit without breaking the feasibility condition. In summary, we conclude that we can increase \( r_i \) by one unit without decreasing any other flow rate to a value lower than the new value of flow \( i \) while maintaining feasibility. This contradicts the discrete maxmin definition.

\[ \]

The previous theorem is the discrete version of the bottleneck condition in the continuous domain presented in [BER84]. Its usefulness remains in that it provides a tractable way to check whether a rate allocation is discrete maxmin. However, it does not indicate how to obtain this rate allocation. Later in this chapter, we will show a procedure to find a discrete maxmin solution.

As mentioned before, the discrete maxmin definition can be interpreted as a way to unify both continuous and discrete domains. However, the discrete bandwidth allocation
has two properties that differ from those in the continuous case. In actuality, these properties make the discrete case a harder to solve problem.

Property 6.1. Non-Uniqueness. The solution of the discrete maxmin problem may not be unique.

Proof. Notice that the discrete maxmin problem is not a LOP, since its feasible set is not convex. Hence, Property 2.3 (uniqueness property) does not apply. To prove non-uniqueness, consider the network in Figure 3. As we showed before, such network has more than one solution.

Property 6.2. Lexicographic Optimal Solution. Any lexicographic optimal solution in the discrete domain is discrete maxmin but not vice-versa.

Proof. Let \( r \) be the lexicographic optimal solution of a network \( N \) and suppose that \( r \) is not discrete maxmin. Then, there must exist a rate vector \( r' \) with an index \( i \) such that \( r'_i > r_i \). Furthermore, there must exist no index \( j \) such that \( r'_j < r_j \) and \( r'_j < r'_i \). This implies that there must exist no index \( j \) such that \( r'_j < r_j \) and \( r'_j \leq r_i \). Now this means that \( r' \) is larger in the lexicographic sense than \( r \), which is a contradiction.

In order to see that not all discrete maxmin solutions are lexicographic we provide a new example. Let us consider the network in Figure 4. It can be seen that the set of discrete maxmin solutions is \{ \( r_1 = [1, 2, 4] \), \( r_2 = [2, 1, 3] \)\}. However, since \( [1, 2, 3] < [1, 2, 4] \)
2, 4] we conclude that \( r_1 \) is the actual lexicographic optimal solution. In this case, \( r_2 \) is a discrete maxmin rate solution but is not lexicographically optimal.

Because in the continuous domain the feasible set is convex and compact, every maxmin solution is also the lexicographic optimal solution (Lemma 2.2). However, such property does not hold in the discrete case. This has direct implications for protocol design. While in the continuous domain the same algorithm will find both the maxmin and the lexicographic optimal solutions, in the discrete domain a general maxmin protocol might not necessarily find a lexicographic optimal solution. Furthermore, one can also argue that the lexicographic optimality condition is in a sense more "maxmin" than the weakened discrete maxmin condition. To understand this statement, we can compare our problem to a classic optimization problem. Usually, when looking for the maximums of a function, we may find the set of local maximums. In our case, this is the set of discrete maxmin rates. Any of these maximums satisfies the optimal condition “gradient equal to zero”; while in our case any discrete maxmin rate satisfies the bottleneck optimality condition. However, we can also find the absolute (global) maximum among the set of local maximums. This is equivalent in our problem to find the
lexicographic optimal solution. In sum, lexicographic optimal solutions are the preferred solutions.

While we have shown the importance of finding the lexicographic optimal solution, the following property proved in [SAS00] shows that the computation of a lexicographic optimal solution is NP-complete.

**Property 6.3. NP-completeness.** The problem of finding the lexicographic optimal solution for the discrete case is NP-complete.

*Proof.* For a rigorous proof, refer to [SAS00].

This result tells us that finding the lexicographic solution can be hard. In the remaining work of this chapter, we will derive a heuristic that aims to find the lexicographic optimal solution in the discrete domain. We first begin by presenting an algorithm that computes an arbitrary discrete maxmin rate. Then, by adding the heuristic, we will be able to compute a solution that is expected to be near the lexicographic optimal solution.

### 4 d-CPG Algorithm

The proposed algorithm, the *d-CPG algorithm*, is based on the concept of CPG graph introduced in previous chapters. Figure 5 and Figure 6 present the pseudo-code. First, let us introduce two more definitions.

**Definition 6.5. Direct related links.** Two links $i$ and $j$ are said to be direct related if there exists a flow that goes through them.
Definition 6.6. **Indirect related links.** Two links \( i \) and \( j \) are said to be indirect related if there exists a link \( k \) such that links \( i \) and \( k \) are direct related and links \( j \) and \( k \) are direct related.

\[
\text{d-CPG Algorithm}(N)
\]

**Input parameters:**

\( N: \) a network

**Output parameters:**

\( \{r_i\}: \) set of discrete maxmin rates

\( \{R_u\}: \) set of advertised rates

1. \( L = 1 \);

2. For each link \( u \) in \( N \), compute \( R_u = \frac{W_u}{|V_u|} \);

3. For each link \( u \) such that \( \min\{ R_u, R_{u'} \} \) \( u \) and \( u' \) are direct or indirect related \( \) and \( W_u - \| R_u \| V_u \| \min\{ W_u - \| R_u \| V_u \| \} \) \( u \) and \( u' \) are direct or indirect related, and \( \| R_u \| = \| R_{u'} \| \) do the following (if two or more different links that are direct or indirect related satisfy the previous condition, arbitrarily choose one):

   3.1. \( \{r_i \mid i \in V_u\} = \text{ComputeRatesHeuristicGeneric}(N, u) \);

   3.2. Remove from the network link \( u \) and any flow crossing this link;

   3.3. For each link still in the network, reduce its capacity by the bandwidth assigned to the flows crossing it and that were deleted in step 3.2;

4. If the set of flows in \( N \) is not empty, do \( L = L + 1 \) and go to 2;

---

Figure 5 d-CPG procedure (see Matlab function CPG() in Appendix)

**ComputeRatesHeuristicGeneric** \((N, u)\)

**Input parameters:**

\( N: \) a network

\( u: \) a link in network \( N \)

**Output parameters:**

\( \{r_i \mid i \in V_u\} \) set of discrete rates for flows crossing link \( u \)

1. \( r_i = \frac{W_u}{|V_u|} \), for any \( i \in V_u \);
2. Choose an arbitrary set of $W_u = \left\lfloor \frac{W}{\|V_u\|} \right\rfloor |V_u|$ flows crossing link $u$ and add one unit to their rate allocation;

Figure 6 ComputeRatesHeuristicGeneric procedure (see Matlab function RATEsinglelink() in Appendix)

The d-CPG Algorithm is a global parallel algorithm. Instead of identifying and solving one bottleneck at a time, it speeds up the computation by identifying at each iteration the set of independent bottlenecks that can be resolved in parallel at the same time (step 3).

Let us now see an example of how the d-CPG algorithm is executed. Consider the network in Figure 7. At the first iteration, we have that $R_1 = 1.5$, $R_2 = 2$, $R_3 = 2$, and $R_4 = 5$. Since the condition at step 3 is only satisfied by link 1, steps 3.1 to 3.3 are only executed for this link. At step 3.1, we will first assign 1 wavelength to both flows A and B and then assign an extra wavelength to any of them. Suppose we choose flow A, then we have $r_A = 2$ and $r_B = 1$. At step 3.2, we remove link 1 and flows A and B from the network. Step 3.3 reduces the link capacities for the remaining links so that $W_2 = 4$, $W_3 = 5$, and $W_4 = 10$. After this, we loop back to step 2 where we compute the new values for the parameter R: $R_2 = 2$, $R_3 = 2.5$, and $R_4 = 5$. In this case, link 2 is the only link satisfying condition 3. At step 3.1 we compute $r_D = r_E = 2$. Steps 3.2 and 3.3 will remove link 2 and flows D and E from the network and compute the new set of capacities: $W_3 = 3$ and $W_4 = 8$. Then, since the network is still not empty, we loop back once again to step 2. After this step, we have $R_3 = 3$ and $R_4 = 8$. At this point, both links 3 and 4 satisfy the condition in step 3 so that steps 3.1 and 3.3 can be executed independently for both of them using parallel processing. By doing so, at the end of step
3.3 we have that $r_c = 3$ and $r_f = 8$, and links 3 and 4 and flows C and F are removed from the network. Finally, the algorithm terminates since the condition at step 4 is not met. In summary, the output rate assignment of the d-CPG algorithm is $r_A = 2, r_B = 1, r_C = 3, r_D = 2, r_E = 2$ and $r_f = 8$.

Figure 7 Network example for d-CPG algorithm

Lemma 6.2. Monotonic rate behavior. Let $R_u^n$ denote the value of $R_u$ at iteration $n$ of the d-CPG algorithm, if link $u$ has still not been removed from the network. Then, $\begin{bmatrix} R_u^1 \\ R_u^2 \\ \vdots \end{bmatrix} \leq \begin{bmatrix} R_u^l \\ \vdots \end{bmatrix}$, where $l$ is the level in the d-CPG algorithm where link $u$ is removed.

Proof. Consider any iteration $n$ such that $1 \leq n < l$. Suppose also that $\Phi = \{f_1, f_2, \ldots, f_m\}$ is the set of flows crossing link $u$ that were removed at iteration $n-1$. In the forthcoming notation, we add a super-index to identify the iteration number. Note that the proof also works for the case $m = 0, \Phi = \emptyset$. Because the minimization at step 3 of d-CPG algorithm is done through both direct and indirect related links, it must be true that all of these $m$ flows were removed from the same link at iteration $n-1$. Let us call this link $u'$. Now from the same step 3, we have that $\begin{bmatrix} R_u^{n-1} \\ \vdots \end{bmatrix} \leq \begin{bmatrix} R_u^{n-1} \\ \vdots \end{bmatrix}$. In addition, if $\begin{bmatrix} R_u^{n-1} \\ \vdots \end{bmatrix} = \begin{bmatrix} R_u^{n-1} \\ \vdots \end{bmatrix}$
then it is true that $W_u^{-1} - \left\lfloor R_u^{-1} \right\rfloor |V_u^{-1}| < W_u^{-1} - \left\lfloor R_u^{-1} \right\rfloor |V_u^{-1}|$. Let us assume first that $\left\lfloor R_u^{-1} \right\rfloor < \left\lfloor R_u^{-1} \right\rfloor$, then,

$$R_u^n = \frac{W_u^{n-1} - \sum_{i=1}^{m} f_i}{|V_u^{n-1}| - m} \geq \frac{W_u^{n-1} - m \left( \left\lfloor R_u^{n-1} \right\rfloor + 1 \right)}{|V_u^{n-1}| - m} \Rightarrow R_u^n \geq \frac{W_u^{n-1} - m \left\lfloor R_u^{n-1} \right\rfloor}{|V_u^{n-1}| - m} = R_u^{n-1} \Rightarrow \left\lfloor R_u^n \right\rfloor \geq \left\lfloor R_u^{n-1} \right\rfloor - 1$$

Finally, let us assume that $\left\lfloor R_u^{-1} \right\rfloor = \left\lfloor R_u^{-1} \right\rfloor$ and $W_u^{-1} - \left\lfloor R_u^{-1} \right\rfloor |V_u^{-1}| < W_u^{-1} - \left\lfloor R_u^{-1} \right\rfloor |V_u^{-1}|$. Then,

$$R_u^n = \frac{W_u^{n-1} - \sum_{i=1}^{m} f_i}{|V_u^{n-1}| - m} \Rightarrow R_u^n = \frac{W_u^{n-1} - \left( W_u^{n-1} - \left\lfloor R_u^{n-1} \right\rfloor |V_u^{n-1}| \right) \left( \left\lfloor R_u^{n-1} \right\rfloor + 1 \right) - m \left( W_u^{n-1} - \left\lfloor R_u^{n-1} \right\rfloor |V_u^{n-1}| \right)}{|V_u^{n-1}| - \left( W_u^{n-1} - \left\lfloor R_u^{n-1} \right\rfloor |V_u^{n-1}| \right) - m \left( W_u^{n-1} - \left\lfloor R_u^{n-1} \right\rfloor |V_u^{n-1}| \right)}$$

using Property 3.4 statement 3 we have that,

$$R_u^n \geq \frac{W_u^{n-1} - \left( W_u^{n-1} - \left\lfloor R_u^{n-1} \right\rfloor |V_u^{n-1}| \right) \left( \left\lfloor R_u^{n-1} \right\rfloor + 1 \right)}{|V_u^{n-1}| - \left( W_u^{n-1} - \left\lfloor R_u^{n-1} \right\rfloor |V_u^{n-1}| \right)}$$

(6.3)

using again the same property and since $W_u^{-1} - \left\lfloor R_u^{-1} \right\rfloor |V_u^{-1}| < W_u^{-1} - \left\lfloor R_u^{-1} \right\rfloor |V_u^{-1}|$, 

145
\[
R_u^n \geq \frac{W_u^{n-1} - \left\lfloor \frac{W_u^{n-1} - \lfloor R_u^{n-1} \rfloor V_u^{n-1}}{\lfloor V_u^{n-1} \rfloor - \lfloor W_u^{n-1} - \lfloor R_u^{n-1} \rfloor V_u^{n-1} \rfloor} \rfloor + 1}{\lfloor V_u^{n-1} \rfloor - \lfloor W_u^{n-1} - \lfloor R_u^{n-1} \rfloor V_u^{n-1} \rfloor}
\] (6.4)

Now expanding and simplifying the expression one can see that,

\[
\frac{W_u^{n-1} - \left\lfloor \frac{W_u^{n-1} - \lfloor R_u^{n-1} \rfloor V_u^{n-1}}{\lfloor V_u^{n-1} \rfloor - \lfloor W_u^{n-1} - \lfloor R_u^{n-1} \rfloor V_u^{n-1} \rfloor} \rfloor + 1}{\lfloor V_u^{n-1} \rfloor - \lfloor W_u^{n-1} - \lfloor R_u^{n-1} \rfloor V_u^{n-1} \rfloor} = \lfloor R_u^{n-1} \rfloor
\] (6.5)

Therefore,

\[
\lfloor R_u^n \rfloor \geq \lfloor R_u^{n-1} \rfloor
\] (6.6)

and the lemma must hold.

\[\text{Theorem 6.2. d-CPG convergence.}\] The d-CPG algorithm computes a discrete maxmin solution.

\[\text{Proof.}\] To proof the theorem we will show that when the algorithm finishes then all the flows must be bottlenecked at some link.

Let \(i\) be an arbitrary flow that is removed at an arbitrary iteration \(n\) and let \(u\) be the link that is removed at the same iteration and that is crossed by flow \(i\). Note first that link \(u\) must be saturated when the d-CPG algorithm terminates because of the reality that it has been removed from the network. Now we have that \(r_i = \lfloor R_u^n \rfloor\) or \(r_i = \lfloor R_u^n \rfloor + 1\). Let also \(j\) be an arbitrary flow crossing link \(u\) that was removed at iteration \(n^*\) with \(n^* < n\). From
step 3 in the d-CPG algorithm, we have that \( R_u^{"n"} \leq R_u^{"n"} \) and from Lemma 6.2 we have that \( R_u^{"n"} \leq R_u^{"n"} \). In addition, we have that \( r_j = R_u^{"n"} \) or \( r_j = R_u^{"n"} + 1 \). Therefore, it is true that \( r_j \geq r_j - 1 \). Now note that flow \( i \) satisfies the three conditions in Definition 6.4.

Therefore, flow \( i \) is bottlenecked at link \( u \).

§

5 Heuristic (LEX) for the d-CPG Algorithm

One manner to work around NP-complete problems is to provide heuristics that, while reducing the computational cost, find the solution in most of the cases, and when not, the solution found is close to the optimal one. In this section we will show that the d-CPG algorithm allows us to incorporate a family of heuristics that can approximate convergence to optimal solutions. We will also provide a heuristic and we will test its accuracy through numerical experimentations.

The d-CPG algorithm presented in section 4 finds one arbitrary vector rate out of the set of discrete maxmin solutions. The arbitrariness of the solution comes from two instances: (1) in step 3 of the algorithm, if two or more different links satisfy the condition at that step and they are direct or indirect related, we arbitrarily choose one of them; (2) in step 2 of the function ComputeRatesHeuristicGeneric we carry out a random selection of the flows for which an extra unit of wavelength is assigned. Now note that by carefully choosing these flows, one can come up with several heuristics that may provide a better solution in the lexicographic sense. The following pseudo-code introduces a heuristic that will demonstrate a high ratio of success in identifying the optimal solution.
This piece of pseudo-code replaces the function \textit{ComputeRatesHeuristicGeneric} in the \textit{d-CPG} algorithm.

\begin{verbatim}
\textbf{ComputeRatesHeuristicLEX (N, u)}

\textbf{Input parameters:}
\begin{itemize}
  \item \textit{N}: a network
  \item \textit{u}: a link in network \textit{N}
\end{itemize}

\textbf{Output parameters:}
\begin{itemize}
  \item $\{ r_i \mid i \in V_u \}$ set of discrete rates for flows crossing link \textit{u}
\end{itemize}

\begin{enumerate}
  \item $r_i = \left\lfloor \frac{W_u}{|V_u|} \right\rfloor$, for any $i \in V_u$;
  \item For any link \textit{k}, subtract from $W_k$ the bandwidth assigned to its crossing flows in previous step 1 and remove these flows from $V_k$;
  \item For each flow \textit{i} crossing link \textit{u}, compute the following vector:
    \begin{align*}
      \omega_i(k) = \begin{cases}
        W_k / |V_k| & \text{if flow } i \text{ crosses link } k \\
        \infty & \text{else}
      \end{cases}
    \end{align*}
  \item Choose a set of $\left\lfloor \frac{W_u}{|V_u|} \right\rfloor$ flows crossing link \textit{u} such that the increasing permutation of their $\omega$ vector is maximal in lexicographic order. For the selected flows, add one unit to their rate allocation;
\end{enumerate}

\textit{Figure 8 ComputeRatesHeuristicLEX procedure (see Matlab function RATEsinglelink() in Appendix)}
\end{verbatim}

The idea behind this heuristic is that we should give the extra unit of bandwidth to those flows that have less probability to bottleneck the network. For each flow that is to be considered (those crossing link \textit{u}), we compute the available bandwidth in each of the links that it crosses. Dividing this bandwidth among the set of flows currently crossing the link provides us a way to intuitively know the chances for this link to become a bottleneck. The smaller this number, the more chances to become a bottleneck. Therefore, for each flow crossing link \textit{u}, we get a vector of rates $\omega$. To choose the flows that have less probability to bottleneck the network, we can take again several
approaches. We can choose some vector ordering and then get those flows that have maximum $\omega$ with respect to this ordering. In our approach, we chose the lexicographic ordering as the way to order these vectors.

6 Numerical Experimentations

In this section we evaluate the efficiency of our heuristic. Figure 9 shows our experimental set-up. We have implemented two algorithms. In one, we use the $d$-CPG algorithm with our heuristic $LEX$. In the other, we implement a modified version of the $d$-CPG algorithm where, instead of finding only one solution, the output of the algorithm is the entire set of discrete maxmin solutions. We then do an exhaustive search in this set and find the lexicographic solution. This rate vector is then compared to the output of the $d$-CPG with heuristic $LEX$. For all the networks that we will test, we will answer two questions: is the solution of our heuristic equal to the lexicographic optimal solution? If not, how far is it from the lexicographic optimal solution?

Figure 9 Experimental set-up
The network generator consists of an algorithm that generates all possible network topologies up to some number of links and nodes. In particular, we generate all possible networks such that $|\Gamma| + |\Omega| \leq 10$, where $\Gamma$ and $\Omega$ are the set of links and the set of flows of the network, respectively. We also fix the amount of bandwidth available at each link to be uniformly distributed with increments of 5 units of bandwidth. In particular, the set of link capacities is defined to be \{10, 15, 20, ..., 5 \| \Gamma \| + 5\}.

Table 1 shows the number of counterexamples found, where a counterexample is a network for which our heuristic does not converge to the lexicographic optimal solution. The cells in this table have two numbers: the first is the number of counterexamples found and the second is the total number of networks for that particular number of flows and links. For example, considering the set of networks with 4 links and 5 flows, we have that our heuristic is successful in 11,156 from the 11,166 possible networks. Adding up all the numbers, the total number of generated networks is 1,084,112, with a total number of 2,069 counterexamples. Therefore, in 99.8% of the generated networks, our heuristic finds the lexicographic optimal solution.

<table>
<thead>
<tr>
<th>Number of links $\rightarrow$ Number of flows $\downarrow$</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>0 / 5</td>
<td>0 / 22</td>
<td>0 / 92</td>
<td>0 / 376</td>
<td>0 / 1520</td>
</tr>
<tr>
<td>3</td>
<td>0 / 9</td>
<td>0 / 74</td>
<td>0 / 596</td>
<td>0 / 4776</td>
<td>0 / 38,224</td>
</tr>
<tr>
<td>4</td>
<td>0 / 14</td>
<td>0 / 195</td>
<td>7 / 2,850</td>
<td>22 / 43,316</td>
<td>717 / 674,344</td>
</tr>
<tr>
<td>5</td>
<td>0 / 20</td>
<td>0 / 441</td>
<td>10 / 11,166</td>
<td>1074 / 313,004</td>
<td>-</td>
</tr>
<tr>
<td>6</td>
<td>0 / 27</td>
<td>2 / 896</td>
<td>237 / 37,836</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

*Table 1 Number of counterexamples for networks with $|\Gamma| + |\Omega| \leq 10$ and with link capacities \{10, 15, 20, ..., 5 \| \Gamma \| + 5\}*

The second question we want to answer is how far is the output of our heuristic from the optimal solution when both happen to be different. Here we use the ordinary $\ell - 1$
norm: $\|x\|_1 = \sum_i |x_i|$ to define the distance of two vectors. Note that with this definition and considering that the vectors belong to the $|\Omega|$-dimensional field of positive integer numbers, the minimal distance between two different vectors is 1. Intuitively, if our first goal of finding the lexicographic order solution fails, then our second best choice should be a discrete maxmin rate that is one unit far from the optimal one.

Another nice result appears when computing the distance between our heuristic solution and the lexicographic optimal solution when they are different. Among all the 2,069 counterexamples found, 2,065 happen to be at minimal distance from the optimal solution (i.e. distance equal to 1) and 4 are at distance 2. In other words, when it fails, the solution of our heuristic is at minimal distance from the lexicographic optimal solution in 99.8% (2,065/2,069) of the cases.

Figure 10 shows the four network topologies that form the exception of not being in minimal distance with respect to the optimal solution.

*Figure 10 Only 4 counterexamples are found whose distance to the lexicographic optimal solution is more than 1 among the set of more than 1 million tested networks with $|\Gamma| + |\Omega| \leq 10$*
Let us now see why our heuristic is not able to succeed for some specific networks. As an example, we will follow the execution of the \(d\)-CPG algorithm for the network in Figure 10a. Starting at the first iteration, after step 2 we have that \(R_1 = 10, R_2 = 7.5, R_3 = 10, R_4 = 12.5, R_5 = 15\). Only link 2 satisfies the condition in step 3 so steps 3.1 to 3.3 are only executed for this link. When executing the heuristic, we have that initially flows B and E are given 7 wavelengths. In addition, when executing the third step in the heuristic we obtain \(\omega_b = [\infty, \infty, 13, \infty, \infty]\) and \(\omega_e = [\infty, \infty, \infty, 18, 23]\). Now since the lexicographic ordering of \(\omega_e\) is bigger than that of \(\omega_b\), our heuristic will give the remaining wavelength in the link to flow E. Then at the end of the first iteration we have \(r_B = 7\) and \(r_E = 8\). The second iteration is straightforward since the remaining network consists of unconnected links crossed by one single flow. As a result, our heuristic will return the rate assignment \(r_A = 10, r_B = 7, r_C = 17, r_D = 22\) and \(r_E = 8\). The actual lexicographic solution is \(r_A = 10, r_B = 8, r_C = 18, r_D = 23\) and \(r_E = 7\), showing that for this case our heuristic cannot converge to the optimal solution.

Let us now analyze the reason for this result. In essence, our heuristic claims that an increase of the rate of a flow that crosses links that have more available bandwidth is less probably to saturate a network than an increase of a flow that crosses links that have less available bandwidth. In our example, we have to choose between increasing the bandwidth of flow B or that of flow E. Intuitively, \(\omega\) vectors tell us that flow E is going through a part of the network that looks less bottlenecked than the part of the network crossed by flow B, since \(\partial_E > \partial_B\). Hence, the heuristic chooses to assign the extra wavelength to flow E. However, in this special case, it turns out that link 3 is never a
bottleneck, even though by the time our heuristic has to make a decision it seems to be the next important bottleneck in the network. This means that the number of available wavelengths at link 3 is going to be underutilized. For convergence to the lexicographic optimal solution, we should try to maximize the bandwidth of those flows that go through link 3. In other words, we should assign the extra wavelength to flow B.

The key issue of any heuristic approach is that of identifying those links that within the lexicographic optimal solution are less utilized. Obviously, it is not possible to dispose of this information since it requires the knowledge of the solution itself. The problem allows for the implementation of many optimization techniques in the discrete domain. For example, sometimes it is not necessary to reach the end of the algorithm in order to realize that we are not going to find the optimal solution. In this case, one could use a backtracking scheme and undo the last iterations to continue through another branch in the tree of possible solutions. Such approach could possibly guarantee convergence to the lexicographic optimal solution. The drawback is that by doing backtracking, it is not possible to bound the cost of our algorithm. In the worst case, we require to do an exponential search through all the branches in the solution tree. A good property of our heuristic is that we can bound the computational cost to a relatively small number of iterations while still provide a high probability of finding the optimal solution.
Chapter VII

CONCLUSIONS AND FUTURE RESEARCH

“[…] Everywhere I go, I hear the same thing.” He stands up, goes to the board, and mimicking a first grade teacher he intones “A process… of… on-going… improvement.”


1 Contributions

In what follows, we summarize the contributions of this dissertation.

- **Unimodality and equivalence lemma.** In chapter one we prove that the operation of ordering in increasing permutation all the elements in a convex set preserves the convexity properties of its lexicographic ordering. This result is used to prove the uniqueness of the lexicographic solution. The same result is crucial in order to provide sufficient conditions to relate Bertsekas’s [BER92] maxmin optimality condition and the lexicographic optimal solution.

- **Bottleneck optimality condition.** The bottleneck optimality condition presented here is a generalization of the classic maxmin optimality condition presented by previous authors
[BER92]. The generalization comes in that ours supports the assurance of a minimal rate service for each user (Quality of Service).

- *Projection optimality condition*. This theorem provides a new maxmin optimality condition derived from the concept of advertised rate. Its simplicity allows us to develop a scalable protocol in terms of switch complexity.

- *Precedent link relationship and bottleneck ordering theory*. The existence of a precedent link relationship is important in order to derive a new maxmin bottleneck ordering theory. The bottleneck ordering theory provides a useful framework to: (1) analyze the bottleneck structure of the network and (2) measure the complexity of a maxmin algorithm.

- *Bottleneck and projection optimality conditions for multicast networks*. These theorems generalize our previously mentioned optimality conditions to the multicast case. They provide multicast optimality conditions with support for multi-rate trees and minimal flow rates in a per-destination basis.

- *Definition of discrete maxmin*. Classic maxmin definition cannot be used when the flow rates are discrete. A new definition of maxmin rate allocation is provided for the case of discrete rates. Our definition was found to be independently formulated by Sarkar [SAS00]. While Sarkar used it to solve the multi-rate multicast allocation problem, we applied this result to the wavelength distribution problem in optical networks. In our theory, we prove that such definition, when used in the continuous space, is equivalent to the classic maxmin definition. This allows for a unified maxmin framework.
- **Heuristic to find discrete lexicographic solutions.** We provide a basic algorithm that can derive into a family of heuristics to identify lexicographically optimal or almost-lexicographically optimal solutions in the discrete domain.

- **Fluid model.** A fluid model representation of the switch algorithm is derived that allows us to solve the switch maxmin problem with minimal rate guarantees using $\log(N)$ iterations, where $N$ is the number of flows crossing the switch. Such complexity cost improves previous authors’ that had at their best a complexity cost of $O(N)$ [CHA95-HOU98-LON99]. It is claimed that this result makes the distribution protocol scalable.

- Bi-directional minimization and transient oscillation freedom. A new signaling protocol is proposed that reduces by half the convergence time of previous authors’ [CHA95-HOU98-LON99-KAL97]. The same signaling protocol proves to reduce the amount of rate oscillations during transient states.

2 **Future Research**

Traditionally, the problems of flow control and routing have been treated separately. Consider the case of BGP (routing protocol [PER99]) and TCP. Neither the first is aware of the rate congestion nor the latter knows about the routes currently defined in the network. In calculus, it is well known that the optimization of a function carried out by the optimization of its parts leads to sub-optimal solutions. In fact, such sub-optimality has proven in several examples to notably downgrade the performance of the Internet (see the fish problem in [WAN01]). Therefore, from an overall efficiency standpoint it would be beneficial to jointly solve the routing and flow control problems together. Our
work has assumed that routes are already established when the maxmin computation starts (e.g. by the means of a separate routing protocol). A future line of research could be the introduction of routes as an optimization parameter into the maxmin problem. We define the solution of this problem as follows:

*Definition 7.1. Lexicographic joint routing and flow control solution (Lexicographic joint solution).* Let $N$ be a network with undefined routes. Considering all possible routes, let $X$ be the set of feasible rates for each user in the network. Then the solution to the **lexicographic joint routing and flow control problem** is that element $x$ in $X$ that is the largest in the lexicographic sense among all the elements in $X$. For the sake of simplicity, we also refer to this solution as the **lexicographic joint solution**.

While protocols in the Internet are not implemented using the joint approach, there exist some problem examples in the literature that have been solved by finding both the routes and the rates of each route. The most commonly known is the max-flow problem [BER97], which defines the optimal solution as that set of routes and routes-rates that provide the maximum aggregated throughput. However, while such solution is optimal in terms of network utilization, it does not take into consideration fairness among the users. If starvation scenarios are to be avoided, then fairness becomes a crucial aspect. Hence, the interest in studying the lexicographic joint problem.

Lexicographic optimization imposes and additional challenge because, as economist Debreu proved, there exists no continuous function representation of the lexicographic criterion [DEB54]. The challenge comes in that a new set of optimization methods may need to be derived.
Bibliography


function [A,S,CPG] = CPG(C,VCM,p,method)
% C: Capacity Vector (each component is a link capacity)
% VCM: Network topology matrix (VCM(i,j)=1 means VCi crosses link j)
% p: fairness+feasibility parameters
% method: "ordinary", "tax", "penalty", "constrained", "discrete-1" (generic),
% "discrete-2" (lex heuristic)
% A: GCM rate allocation
% Link Satisfaction
% CPG: CPG Graph
% Input Sample:
% C=[150]
% VCM=[1;1;1]
% [A,S,CPG]=CPG(C,VCM,[1,1,1;100,40,0;200,200,200],"constrained")
% Jordi Ros Giralt - 2.99
% NETROL Group. University of California Irvine
% % initialization
L = 1:size(VCM,2); % Set of Links
VC = 1:size(VCM,1); % Set of VC's
A = zeros(1,size(VCM,1)); % Set of rates
S = zeros(1,length(C)); % Set of advertised satisfactions
CPG = zeros(size(VCM,2)); % CPG
VCMP = VCM; % Copy of VCM
level = 0; % Current level of CPG
parallel = 0; % Solve in parallel
% % algorithm
% while size(L) ~= 0
level=level+1;
% Solve the set of single links
for j = L

\[ S(j) = SATsinglelink(C,VCMp,p,VC,j,method); \]

end

\% Find the most bottleneck disjoint links
if ~parallel
    minim = 999999; 
    for \( k = L \)
        if \( S(k) < \text{minim} \)
            \( i = k; \)
            \( \text{minim} = S(k); \)
        end
    end
    \( CPG(i,i) = \text{level}; \)
A = RATEsinglelink(C,VCMp,p,VC,A,S,i,method);
else
    for \( i = L \)
        \( \text{minimum} = 1; \)
        for \( j = \{1:\text{find}(L==i)-1,\text{find}(L==i)+1:\text{length}(L)\} \)
            if \( \text{VCMp}(i,j) \& S(i)>S(j) \)
                \( \text{minimum} = 0; \)
            end
        end
        if minimum
            \( CPG(i,i) = \text{level}; \)
            A = RATEsinglelink(C,VCMp,p,VC,A,S,i,method);
        end
    end
\end
VC = VC([1:find(VC==j)-1,find(VC==j)+1:length(VC)]);
end

function S = SATsinglelink(C,VCMp,p,VC,j,method);

switch lower(method)
  case 'ordinary',
      S = C(j)/sum(VCMp(:,j));
  case 'discrete-1',
      S = C(j)/sum(VCMp(:,j));
  case 'discrete-2',
      S = C(j)/sum(VCMp(:,j));
  case 'constrained',
      S = C(j)/sum(VCMp(:,j));
  case 'tax'
      S = C(j)/sum(VCMp(:,j));
  otherwise, error('Unknown method.');
end

% p is the vector tax
sum = 0;
for i=1:length(VC)
    sum=sum+VCMp(i,j)*p(VC(i));
end
S = C(j)/sum;

function A = RATEsinglelink(C,VCMp,p,VC,A,S,j,method);

% N=0;
% M=0;
for i=1:length(VC)
    N=N+VCMp(i,j);
    M=M+VCMp(i,j)*p(VC(i));
end
if M ~=0 & M ~=N
    S=(-M+sqrt(M^2+4*C(j)*M*(N-M)))/2/M/(N-M);
else
    S=C(j)/N;
end
otherwise, error('Unknown method.');
end

switch lower(method)
    case 'ordinary',
        for i=1:length(VC)
            if VCMp(i,j)==1
                A(VC(i))=S(j);
            end
        end
    case 'discrete-1'
        diff = 0;
        for i=1:length(VC)
            if VCMp(i,j)==1
                A(VC(i))=floor(S(j));
                diff = (diff - A(VC(i)) + S(j));
            end
        end
        diff = round(diff);
        rvc = [];
        % list of removed VCs
        while(diff ~= 0)
            maxim = -1; % available rate of the VC less constrained
            imaxim = -1; % index of that VC
            for i=1:length(VC)
                if (VCMp(i,j)==1 & (isempty(rvc) | ~sum(i == rvc)))
                    minim = 999999999;
                    for k=1:length(S)
                        if (k ~= j & VCMp(i,k)==1 & S(k)<minim)
                            minim = S(k);
                        end
                    end
                    if (minim > maxim)
                        imaxim = i;
                        maxim = minim;
                    end
                end
            end
            A(VC(imaxim)) = A(VC(imaxim)) + 1;
            rvc(length(rvc)+1) = imaxim;
            diff = diff - 1;
        end
    case 'discrete-2'
        diff = 0;
        Nvc = sum(VCMp,1);Cp = C;
        for i=1:length(VC)
            if VCMp(i,j)==1
                A(VC(i))=floor(S(j));
                diff = (diff - A(VC(i)) + S(j));
                Cp = Cp - A(VC(i))*VCMp(i,:);
                Nvc = Nvc - 1*VCMp(i,:);
end
end

diff = round(diff);

rvc = []'; % list of removed VCs
while(diff ~ 0)
    maxim = -1'*ones(1,length(Cp)); % available rate of the VC
    less constrained
    imaxim = -1; % index of that VC
    Sp = Cp./(Nvc+0.0001); % remaining fair share
    for i=1:length(VC)
        if (VCMp(i,j)==1 & (isempty(rvc) | ~sum(i == rvc)))
            Si = VCMp(i,:).*Sp;
            Si = sort(Si + 999999*(Si==zeros(1,length(Si))));
        end
        if(sortrows([maxim;Si]) == [maxim;Si])
            imaxim = i;
            maxim = Si;
        end
    end
    A(VC(imaxim)) = A(VC(imaxim)) + 1;
    rvc(length(rvc)+1) = imaxim;
    diff = diff - 1;
    Cp = Cp - 1*VCMp(imaxim,:);
end

end

case 'tax'
for i=1:length(VC)
    if VCMp(i,j)==1
        A(VC(i))=S(j)*p(VC(i));
    end
end

end
case 'penalty'
    M=0;
    N=0;
    for i=1:length(VC)
        M=M+VCMp(i,j)*p(VC(i));
        N=N+VCMp(i,j);
    end
    for i=1:length(VC)
        if VCMp(i,j)==1
            if p(VC(i))==1
                A(VC(i))=S(j);
            else
                if M ~=0
                    A(VC(i))=S(j)^2*M;
                else
                    A(VC(i))=S(j);
                end
            end
        end
    end

end
case 'constrained'
    Nw = 0;
for i=1:length(VC)
Nw = Nw + VCMp(i,j)*p(1,VC(i));
end
r=zeros(1,length(VC))-2;
done = 0;
while ~done
Nwc = 0;
Fc = 0;
for i=1:length(VC)
    if VCMp(i,j) & (r(i) == p(2,VC(i)) | r(i) == p(3,VC(i)))
        Nwc = Nwc + p(1,VC(i));
        Fc = Fc + r(i);
    end
end
for i=1:length(VC)
    if VCMp(i,j) & (r(i) == p(2,VC(i))) & (r(i) == p(3,VC(i)))
        r(i) = p(1,VC(i))*(C(j)-Fc)/(Nw-Nwc);
    end
end
R=0;
for i=1:length(VC)
    if VCMp(i,j) & (r(i) > p(2,VC(i))) & (r(i) > R*p(1,VC(i))) & (r(i) < p(3,VC(i)))
        R = r(i)/p(1,VC(i));
    end
end
done = 1;
for i=1:length(VC)
    if VCMp(i,j) & (r(i) == min(max(R*p(1,VC(i)),p(2,VC(i))),p(3,VC(i))))
        done = 0;
        r(i) = min(max(R*p(1,VC(i)),p(2,VC(i))),p(3,VC(i)));
    end
end
end
otherwise, error('Unknown method.')