Hierarchical Virtual Paths Allocation in Large-Scale ATM Networks using Noncooperative Game Models

by

Dalia M. B. Fayek

A thesis
presented to the University of Waterloo
in fulfilment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Electrical and Computer Engineering

Waterloo, Ontario, Canada, 2001

©Dalia M. B. Fayek 2001

National Library of Canada
Acquisitions and Bibliographic Services
395 Wellington Street
Ottawa ON K1A 0N4
Canada

Bibliothèque nationale du Canada
Acquisitions et services bibliographiques
395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence

Our file Notre référence

The author has granted a non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.

The author retains ownership of the copyright in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de cette thèse sous la forme de microfiche/film, de reproduction sur papier ou sur format électronique.

L'auteur conserve la propriété du droit d'auteur qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

0-612-65236-X

Canada

The University of Waterloo requires the signatures of all persons using or photocopying this thesis. Please sign below, and give address and date.

Abstract

Broadband Integrated Services Networks (B-ISDN) form the foundation for the modern communication networks due to their ability to transport different types of traffic classes such as voice, video and data. The Asynchronous Transfer Mode (ATM) networks were chosen to be the transmission mechanism of B-ISDN.

The configuration of Virtual Path (VP) connection services plays an important role in the design and operation of large-scale ATM networks. VPs can be regarded as "elastic bands" of reserved bandwidth on the network links. The major research challenge is to account for the fundamental trade-off between the overall network throughput and the processing load on the signaling system. The first being affected by the amount of reserved and hence not fully used capacity; while the increased signaling load is affected by the amount of connections to be processed. Therefore, it is essential to provide an algorithm for VP capacity allocation that achieves an optimal network operating point while guaranteeing the QoS at the call level and satisfies a priori bounds on the processing load of the call processors.

Recently, system-wide, centered control of large-scale networks was recognized to be prohibitive and impractical. A hierarchical structure segments a large network into smaller, manageable entities and provides more flexibility to the changes in the network topology. Techniques for representing and passing information between hierarchical levels are explored. An issue of accuracy of representation arises as the level of information abstraction increases due to the diminished quantity of details broadcast about the status of the network resources between the nodes.

Furthermore, users competing for the network resources can be regarded as a collection of noncooperative players acting in a greedy manner. For this purpose, game theory provides the analytical tools to model their behaviour. In general, a

game with N players, simultaneously deciding on their game strategy, can possess a steady state strategy profile, or equilibrium point known as the "Nash-equilibrium". A Nash-equilibrium is the operating point of the network from which a unilateral deviation does not help any player to improve his performance. Moreover, the Stackelberg game model is used to provide more control to network managers by allowing them to arbitrate the game in a prioritized manner. This is achieved by the process of associating higher priorities to connections generating higher revenue. The game proceeds then between connections of the same priority level, beginning with the highest priority connections deciding on their strategy profile and ending with the lowest priority ones responding by competing on the remaining network resources.

In summary, the objective of this work is to search for the best possible allocation of network resources among the competing users in such a way as to increase the network's revenue while maintaining the Quality of Service requirements of the users. Implementation wise, a coordination between local network managers is the framework within the network layers to achieve this objective. The subdivision of the optimization problem among smaller local managers becomes therefore computationally feasible.

Acknowledgment

First of all, I would like to praise God for giving me the gift of life up to this moment. I would like to thank Him for His gracious and abundant love, and for surrounding me with wonderful people!

Second, I would like to express my gratitude to each member in my committee: Prof. Vladimir Marbukh, who acted as my external examiner, Prof. Shawki Areibi, Prof. Paul Calamai, and Prof. Andrew Heunis for their encouragement and help. In addition, the contribution that Dr. Kenneth Holmström provided added significant value to my work.

Many people were a great source of encouragement to me during this phase in my life, but no words of thanks would be enough to express my gratitude to my supervisors, Prof. George Kesidis and Prof. Anthony Vannelli. Their unwavering support, encouragement and understanding, especially during the tough times in this program, will always be cherished and appreciated. This work would not have come into existence without their contribution.

I would like also to thank my friend, Malathi Francis, who shared most of my years at the University of Waterloo with their peaks and valleys.

It is undeniable that the positive influence of my friends at Church helped me to get through this program. Father Athanasius Iskander's guidance and advice were a constant source of motivation for me. Without any doubt, the baby-sitting days that Maryan Iskander provided were indispensable for the completion of this work. If space were to permit it, I would have liked to thank each one by name!

Finally, my most sincere feelings of gratitude are for my parents, my son Pavly and my husband, Réda. My parents provided the best environment that any person

can thrive in. I thank them for it! I thank my son Pavly for being such a lovely, not so demanding, baby and for giving his mother the opportunity to study during his regular naps!

As for my beloved husband, Réda, he was the most supportive person to me during this whole Ph.D. program duration. His consistent encouragement, unfading help and patience maintained my confidence, especially in those critical times of my sickness. Knowing that he was there provided the security that I needed most. His help, especially after the birth of our son, was indispensable for me to get my work done. I will not be able to thank him enough!

Contents

1 Introduction **1**

1.1 Overview of Current Approaches 3

1.2 Document Layout 8

2 Virtual Paths in ATM Networks **11**

2.1 Loss Networks 11

2.2 Virtual Path Partitioning in ATM Networks 14

2.2.1 Problem Formulation of a VP Partitioning Problem 17

2.2.2 Routing Policies 20

2.3 Noncooperative Game Models 22

2.3.1 Finite Noncooperative Games 25

2.3.2 Infinite Nonzero-sum Games 30

3 Hierarchical Network Management **35**

3.1 Hierarchical Network Structure 36

3.2 VP Partitioning and Routing Process 40

3.2.1 Global Network Management 40

3.2.2 Topology Encoding in Subnetworks 44

3.2.3 Capacity Allocation in Local Subnetworks 46

4 System Implementation **57**

4.1 Implementation of Hierarchical VP Allocation 58

4.1.1 Physical Construction 59

4.1.2 Clustering and Hyper-Network Construction 61

4.1.3 Random Traffic Generation 63

4.2 Data Exchange 64

4.2.1 Data Structures 64

4.2.2 Interface with the Optimizer 67

4.3 Processing of Routed Connections 72

4.4 Implementation Variants 77

4.4.1 The Discrete Optimization Solver: "*glcSolve*" 79

4.4.2 The DIRECT Global Optimization Algorithm 81

4.4.2.1 DIRECT Algorithm in One Dimension 84

4.4.2.2 DIRECT Algorithm in Multiple Dimensions 87

4.4.3 The Continuous Optimization Solver: "*conSolve*" 89

5 Analysis and Experimental Results **95**

5.1 Experimental Comparison 95

5.1.1 First Network 101
5.1.2 Second Network 110
5.1.3 Third Network 120
5.2 Summary of Results 121

6 Summary **123**

6.1 Summary 123
6.2 Summary of Contributions 125
6.3 System Evaluation and Potential Extensions 126

A Classes Implementation **131**

A.1 Network Data Structures 131
A.1.1 Node Class 131
A.1.2 Edge Class 134
A.2 Traffic Data Structure 136

B Examples of Matlab Files **139**

B.1 M-File used for Network Modeling 139
B.2 M-Files For the Objective Functions 147
B.2.1 Objective Function for the Discrete Game Model 147
B.2.2 Objective Function for the Discrete Revenue Model 150

Bibliography **155**

List of Tables

1.1 Comparison of VP allocation algorithms 4

5.1 Traffic Data for Net (1). Net (2) and Net (3). 101

5.2 Analysis for Network (1) and Traffic set (1). 104

5.3 Analysis for Network (1) and Traffic set (2). 105

5.4 Analysis for Network (1) and Traffic set (3). 106

5.5 Analysis for Network (2) and Traffic set (1). 112

5.6 Analysis for Network (2) and Traffic set (2). 113

5.7 Analysis for Network (2) and Traffic set (3). 114

5.8 Analysis for Network (3). 120

List of Figures

1.1 Document Layout. 9

2.1 Iso-cost curves and the corresponding reaction curves for a two-player infinite game. 31

2.2 Nash and Stackelberg equilibria 33

3.1 Hierarchical organization in a large-scale network. 38

3.2 VP allocation and routing for a level-K demand. 42

3.3 Information exchanged between network layers. 43

3.4 Topology encoding by Full-mesh construction. 47

3.5 Example of traffic demands in a layer (1) subnetwork, with $K = 3$. . 50

3.6 Functional description of a local network manager. 52

4.1 Relational Outline of the Simulator Modules 58

4.2 Operation of the VP partitioning and routing processor. 68

4.3 Traffic flow in an intermediate node. 71

4.4 Shubert's algorithm 83

4.5 Computing a lower-bound with centerpoint sampling. 85

4.6 Set of potentially optimal intervals. 86

4.7 Dividing rectangles in the DIRECT algorithm. 90

4.8 Newton's method for minimization. 94

5.1 Network (1), with 16 hypernodes (nodes (10) to (25)). 97

5.2 Network (2). 98

5.3 Network (3). 99

5.4 First and second layers on top of the physical network Net (1) (Figure 5.1). 100

5.5 Throughput values for Net (1) (Equation 5.1) using: (a) the discrete game model, (b) the discrete revenue model and (c) the continuous game model. 108

5.6 Throughput values for Net (1) (Equation 5.2) using: (a) the discrete game model, (b) the discrete revenue model and (c) the continuous game model. 109

5.7 Link Utilization for Top-Down approach when used for Net (1) and under traffic loads: (a) T1, (b) T2 and (c) T3. 110

5.8 Run-Time values for: (a) Discrete Game model, (b) Discrete Revenue model and (c) Continuous Game model. 111

5.9 Throughput values (Equation 5.1) for Net (2) using: (a) the discrete game model, (b) the discrete revenue model and (c) the continuous game model. 116

5.10 Throughput values (Equation 5.2) for Net (2) using: (a) the discrete game model, (b) the discrete revenue model and (c) the continuous game model. 117

5.11 Throughput values for Net (2) when using the Top-Down Approach and under increasing traffic demands. 118

5.12 Link Utilization for Top-Down approach when used for Net (2) and under traffic loads: (a) T1, (b) T2 and (c) T3. 118

5.13 Run-Time values for: (a) Discrete Game model and (b) Discrete Revenue model. 119

5.14 Throughput values for Net (3) (Equation 5.2) 121

Chapter 1

Introduction

Modern computer networks are large-scale systems that carry a wide variety of traffic classes, such as video, audio and computer data, each with its own characteristics and quality of service requirements. Supported applications range from simple file transfers to resource demanding multimedia services such as video-conferencing and network management schemes.

The Asynchronous Transfer Mode (ATM) is the transmission and multiplexing technique used by Broadband Integrated Services Digital Networks (B-ISDN) for the transport of different services. Since B-ISDNs support various classes of multimedia traffic with different bit rates and quality of service requirements, traffic control and resource management are crucial in order to guarantee the desired grade of service. Call admission, input rate regulation, routing and buffer management constitute different mechanisms to control traffic in a B-ISDN. Control can be applied at different levels, e.g. cell level and connection level. ATM networks are connection-oriented by design, since the communication is based on end-to-end call establishment.

Virtual Paths (VP's) are used in ATM networks as a mechanism for traffic control. The VP can be viewed as a pre-established logical route through the network with dedicated bandwidth onto which individual connections, or Virtual Channels (VC's), are multiplexed. VP's considerably simplify the hardware in the transit nodes, provide simple VC admission control and reduce the call set-up delays. However, the use of VP's also increases the call blocking rate and reduces the degree of capacity sharing in communication networks [1, 12, 47].

In the VP allocation domain, the research challenge is to account for the inherent trade-off between the increase in total call blocking due to VP capacity allocation and processing cost due to call set-up delays. The goal is to find the optimal balance in the network operation by minimizing the signaling costs produced by call set-up on one hand, and by accounting for the decrease in links utilization, consequently the network throughput, due to bandwidth reservation on the other hand.

In the network design phase, the VP capacity allocation is an NP-hard optimization problem. A variety of approaches that aim to find the near-optimal VP configuration in ATM networks is found in the literature. Recently, this problem was viewed from a game perspective, where different connection requests are considered as "selfish" users competing for the network resources. The self-optimizing behaviour of the users leads to a dynamic behaviour of the network. Game theory provides the systematic framework to study and understand the behaviour of noncooperative users in communication networks. A game models situations where individuals with different, and at times conflicting, interests interact. The operating points of a noncooperative network are the Nash equilibria of the underlying game, that is, the points where unilateral deviation does not help any user to improve his performance. In the case of games with relative priorities, where one or some users have precedence in selecting their game strategy, the operating point is then called

a Stackelberg equilibrium [7].

In large-scale networks, the information processing becomes unmanageable due to the network size. Conceptually organizing the network into several hierarchical levels has been shown to be beneficial for information handling purposes [17, 70]. Decentralized control algorithms provide the overall network management.

1.1 Overview of Current Approaches

The VP allocation algorithms found in the literature, can be classified as:

- *Synchronous or Asynchronous*

  Synchronous algorithms maintain a fixed VP distribution for a certain *update interval*. These algorithms are based on estimates of the offered load during the coming period. Asynchronous algorithms update the VP capacities in real-time based on the observed demand for call establishment, i.e., at call arrival or departure times [58].

- *Centralized or Decentralized*

  Centralized algorithms are run in one location, typically at the network manager's site and require the collection of up-to-date information from all network nodes. Decentralized algorithms are run in every network switch and information is passed along between neighbour switching nodes.

Table 1.1 points out the characteristics of some of the classical algorithms based on the cost or objective function used, the set of constraints and the number of

	Objective	Constraints	Sync/Async	Cent/Decent	# Services
[1]	min maximum flow	link capacity propagation delay	Sync	Cent	Multiple
[3]	max revenue	link capacity switch signaling call blocking	Async	Cent	Multiple
[6]	min # VPs/connection	link capacity call blocking call set-up delay	Sync	Cent	Multiple
[12]	min max link congestion	link capacity	Sync	Cent	Single
[22]	shortest path	link capacity load on VP	Sync	Cent	Multiple
[25]	min processing cost	link capacity call set-up time	Async	Cent	Single
[47]	min call blocking rate	link capacity #links per VP	Async	Cent	Single
[58]	feasibility	link capacity	Sync	Decent	Single
[66], [67]	max throughput	link capacity call blocking	Sync	Cent	Multiple

Table 1.1: Comparison of VP allocation algorithms

services. The taxonomy of the algorithm whether it is synchronous or asynchronous. centralized or decentralized is also shown.

In some recent work. dynamic routing algorithms that preserve the bandwidth guarantees are studied [33] in a MPLS (multiprotocol label switched) networks. Path selection heuristics are developed based on the idea of deferred loading of certain critical links that are identified as links which. if heavily loaded. would make it impossible to satisfy future demands between certain source-destination pairs. Unlike off-line routing algorithms that require *a priori* knowledge of all tunnel requests that are to be routed. the algorithm presented in [33] is based on the idea that a newly routed tunnel must follow a route that does not "interfere too much" with a route that may be critical to satisfy a future demand.

Some other works used genetic algorithms to solve the VP partitioning problem [62]. Furthermore. as an extension to the already allocated VPs in any given network. some work studied the effectiveness of fairly redistributing the unused reserved capacity to enhance the overall network throughput by maximizing the degree of sharing that was reduced by the VP partitioning itself. while maintaining the benefits of a virtual network [21].

In [14]. a framework is introduced that deals with the load conditions in best-effort networks. like the Internet. to support connections with quality of service guarantees. The quality of service in this context is expressed in terms of the percentage of time the bandwidth allocated to a connection may drop below a certain level or the maximum allowable delay in placing the call. Extensions to the proposed model consider the case of dynamic pricing which allows connections that pay more to get larger shares of the bandwidth.

Resource management in virtual private networks (VPN) is of growing importance. Several current research opt to find practical ways that dynamically establish secured connectivity [15]. Dynamic multicast resource allocation is also widely used [11, 64] where the objective is to design a low cost multicast tree from the source that would provide the QoS levels requested by the receivers.

Numerous work can be found in the literature that address the problem of provisioning the network resources in the best possible way based on a predefined set of constraints, such as the work found in [8, 19, 69].

Bandwidth allocation and QoS guarantees in satellite networks and wireless ATM networks are designed with the same concepts with provisioning for the dynamic nature of the nodes and the corresponding traffic patterns and queueing models [8, 51, 72].

One common feature to all previous approaches is that the dynamics of connections behaviour incorporated in the model formulation used Poisson modeling. Recent work, on the other hand, is based on the assumption that the incoming connections would act in a competitive manner over the network resources and thus make use of game models. The discussion that follows gives a brief overview about these recent works.

In [53], consideration is given to the problem of optimal flow control in a multiclass telecommunications environment where each user (or class) desires to optimize its performance while being fair to the other users (classes). The Nash arbitration scheme from game theory is shown to be a suitable candidate for a fair, optimal operation point in the sense that it satisfies certain axioms of fairness.

A communication network shared by several selfish users is considered in [59]. Each user seeks to optimize his own performance by controlling the routing of his

given flow demand, giving rise to a noncooperative game. The Nash equilibrium of such systems is investigated. For a two-node multiple-link system, the uniqueness of the Nash equilibrium is proved under reasonable convexity conditions. It is shown that this Nash equilibrium point possesses interesting monotonicity properties. For an arbitrary network, the uniqueness of the Nash equilibrium is established under various assumptions [42, 59].

In [52], an application of cooperative game theory to the synthesis of fair call admission controls for multi-service loss networks is presented. The proposed model for evaluation of three different game schemes is based on the value iteration algorithm from Markov decision theory. The arbitration schemes are compared with two traditional call admission objectives, traffic maximization and blocking equalization. The comparison demonstrates that the arbitration solutions provide some attractive fairness features not possessed by traditional objectives, especially in overload conditions.

In [2] the problem of designing an international telecommunication network when multiple countries are involved is studied. Basically, a trade-off between two typical scenarios is assessed based on the fact that most countries will gain by having a truly international network which allows routing via transit countries: this takes advantage of distributed peak traffic hours, and improves the network resource utilization. Whether countries may obtain greater gains by either behaving preemptively or by waiting for the major countries to make their decisions, and then being followers in a Stackelberg game is discussed.

Current research that models the interaction between competing users on network resources as a noncooperative game can be found in [29], [39], [40], [41] and [42]. The resources subject to competition are the link capacities in a Virtual Path allocation game, and traffic routing over the configured network. Issues

such as network pricing are discussed in, for instance, [13] and [44].

The Nash arbitration scheme from cooperative game theory also provides a natural framework to address the allocation of available bandwidth in network links which is network, or "Pareto" optimal and satisfies prespecified notions of fairness. A distributed approach for bandwidth allocation is proposed in [73] based on the Nash arbitration scheme.

Some research addressed the network management in incomplete information circumstances. In [50], a network management framework is proposed to balance the network performance under normal and adverse operational conditions. Both Bayesian techniques and zero-sum game models are used in particular extreme cases where the network environment status is aggregated in *(i)* singleton mutually exclusive partitions of the environment strategy space (Bayesian technique), and *(ii)* the other extreme where the two partitions are the set of all possible values that the environment state variavle can have and the empty set (minimax or zero-sum game between the network and the environment). In [50], the proposed framework allows the network to develop a set of measured responses to possible anomalies by minimizing the average maximum network losses or risks due to uncertainty.

1.2 Document Layout

The document organization is illustrated in Figure 1.1. An introduction to the VP capacity allocation problem was presented in the introductory discussion. A survey of some current related research in the VP dimensioning domain followed in Section 1.1.

Chapter 2 expands the problem description by first describing the loss network

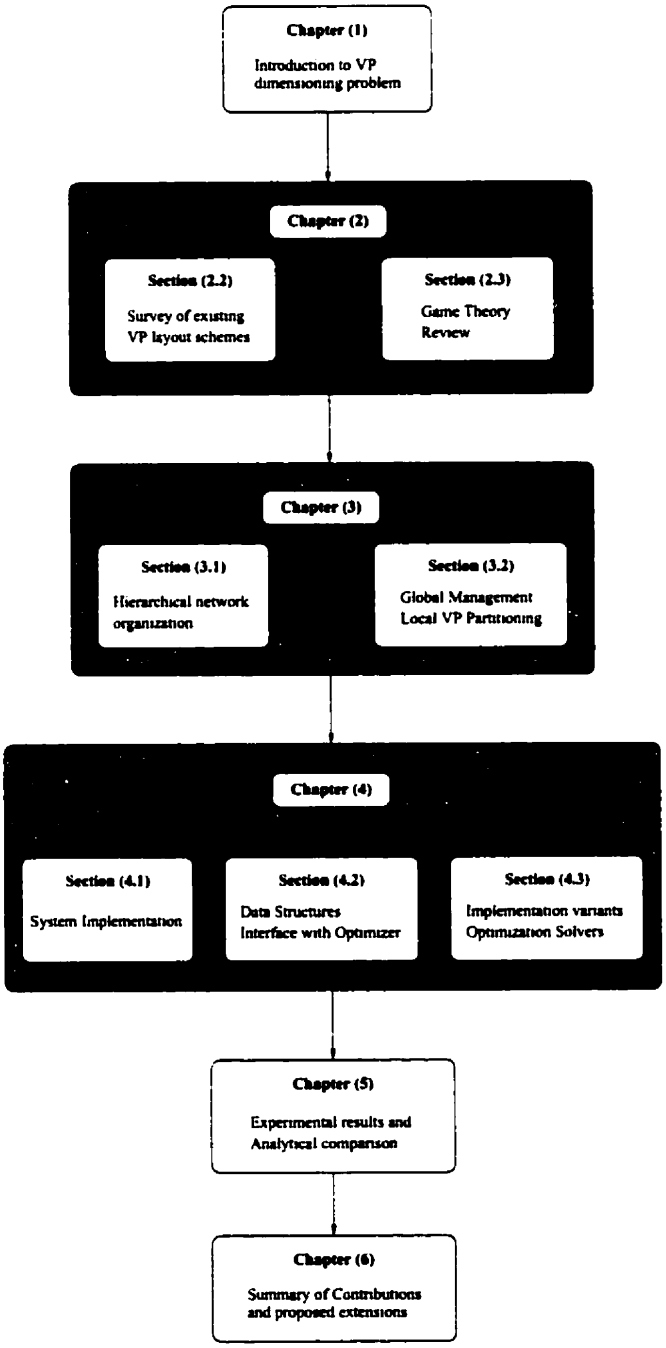


Figure 1.1: Document Layout.

model in Section 2.1. Second, based on that model, Section 2.2 provides the classical formulation of the VP capacity allocation problem with the objective of finding the best network operating point subject to a set of constraints. Section 2.3 discusses the finite and infinite noncooperative games, and the conditions under which Nash and Stackelberg equilibria exist.

In Chapter 3, first the hierarchical structure of the network organization is presented in Section 3.1. Section 3.2 outlines the type of interaction between network management entities distributed among the different hierarchical levels (Section 3.2.1). Further detailed description of the actual mathematical model and the problem formulation is also given (Section 3.2.3) that focuses on a local subnetwork and explains the task of the local network manager in optimizing the VP allocation within its administrative domain.

The actual implementation is discussed in Chapter 4. Section 4.1 starts by explaining the development steps of the current system. Following is an outline of some practical programming issues in Section 4.2. Finally, some modifications to the initial system are suggested in Section 4.4.

In Chapter 5, the developed system is tested and the experimental results are reported. A comparison between the different mathematical models, as well as a comparison between the different optimization modes, i.e., single or multi-level, are studied and analysis is performed using various perfomance measures.

Finally, Chapter 6 contains a summary of the contributions of the present work. Some concluding remarks about the relevant extensions that can be augmented onto the current system together with an assessment of its strengths and weaknesses are discussed.

Chapter 2

Virtual Paths in ATM Networks

Following the introduction presented in Chapter 1, we describe the bandwidth allocation for Virtual Paths problem from a classical point of view. Furthermore, we provide the necessary game theory background that supports our problem formulation that will be discussed in the next chapter.

In this chapter, we start by describing the Erlang formula and loss networks in Section 2.1. Consequently, we give the mathematical model for a typical VP partitioning problem in Section 2.2. Finally, we provide a brief background about game models in Section 2.3

2.1 Loss Networks

In 1917, the Danish mathematician A. K. Erlang studied the problem of calls arriving at a link comprising C circuits as a Poisson process with rate λ. With each call holding a circuit for an independent exponentially distributed time with mean $1/\mu$ and by defining the "traffic intensity" $\rho = \lambda/\mu$. Erlang published his famous

formula.

$$\mathcal{E}(\rho, C) = \frac{\frac{\rho^C}{C!}}{\sum_{n=0}^{C} \frac{\rho^n}{n!}} \tag{2.1}$$

for the loss probability of a telephone system [10]. A call is blocked and lost if all C circuits are occupied, and call holding times are independent of each other and of arrival times, in addition, they are identically distributed with mean $1/\mu$. Erlang obtained formula (2.1) from his development of the statistical equilibrium concept [10].

We now identify Erlang's concept with the stationary measure of a Markov process. Indeed, we also now realize that the result holds under much weaker independence assumptions, for example, the loss probability remains the same if the successive call holding times associated with a given circuit are a dependent stationary sequence. This fact was established by the modern theory of "insensitivity" [34, 63, 71]. In the following overview of loss networks, Equation (2.3) was also shown to be insensitive to the holding time distribution.

The basic model of a loss network can be outlined as follows [36]. Consider a network with J links, with link j comprising C_j circuits. A call on route r uses A_{jr} circuits from link j, where $A_{jr} \in Z_+$. Let $\mathcal{R}$ be the set of possible routes. Calls requesting route r arrive as a Poisson stream of rate λ_r, and Poisson streams are independent. A call requesting route r is blocked and lost if on any link j that is part of route r, $j = 1, 2, \ldots, J$, there are fewer than A_{jr} circuits free. Otherwise, the call is connected and holds A_{jr} circuits on all links on route r for the holding period of the call which is independent of earlier arrival times and holding periods. Holding periods of calls on route r are identically distributed with unit mean, i.e. $1/\mu = 1$, and $\rho = \lambda$.

Let $n_r(t)$ be the number of calls in progress at time t on route r, and define

the vectors $\mathbf{n}(t) = (n_r(t), r \in \mathcal{R})$ and $\mathbf{C} = (C_1, C_2, \ldots, C_J)$. Then the stochastic process $(n(t), t \geq 0)$ has a unique stationary distribution $\pi(\mathbf{k}) = P\{\mathbf{n}(t) = \mathbf{k}\}$ that is given by [63]

$$\pi(\mathbf{n}) = G(\mathbf{C})^{-1} \prod_{r \in \mathcal{R}} \frac{{\rho_r}^{n_r}}{n_r!}, \qquad \mathbf{n} \in \mathcal{S}(\mathbf{C}). \tag{2.2}$$

where the state space $\mathcal{S}(\mathbf{C}) = \{\mathbf{n} \in \mathcal{Z}_+^{|\mathcal{R}|} : \mathbf{An} \leq \mathbf{C}\}$, and $G(\mathbf{C})$ is the normalizing constant or partition function

$$G(\mathbf{C}) = \sum_{\mathbf{n} \in \mathcal{S}(\mathbf{C})} \prod_{r \in \mathcal{R}} \frac{{\rho_r}^{n_r}}{n_r!} \tag{2.3}$$

Most quantities of interest can be written in terms of the distribution (2.2). Let L_r be the stationary probability that a call requesting route r is lost. Since the arrival stream of calls requesting route r is Poisson, then by using PASTA (Poisson Arrival Sees Time Average) [32, 71], an arriving call will be lost if it comes to a full system, i.e. the links $j \in r$ cannot accommodate its request. L_r denotes the blocking probability on route r.

$$1 - L_r = \sum_{\mathbf{n} \in \mathcal{S}(\mathbf{C} - \mathbf{Ae}_r)} \pi(\mathbf{n}) = G(\mathbf{C})^{-1} G(\mathbf{C} - \mathbf{Ae}_r) \tag{2.4}$$

where e_r is a $|\mathcal{R}|$-dimensional vector with all entries equal to zero except the r^{th} entry is equal to unity.

The task of computing the partition function (2.3) for an arbitrary topology was shown to be NP-hard [48, 63]. In fact, the main objective of significant recent work in the network management domain was to find approaches which would avoid the impractical computation of the partition function. One of the alternatives is to use the "*reduced load*" approximation, also known as the "*Erlang fixed point*" approximation.

The Erlang fixed point approximation [36, 63] consists of the following. For a loss network with fixed routing, let $\mathcal{E}_1, \mathcal{E}_2, \ldots, \mathcal{E}_J$ be a solution to the equations

$$\mathcal{E}_j = \mathcal{E}(\rho_j, C_j), \qquad j = 1, 2, \ldots, J \tag{2.5}$$

where

$$\rho_j = \sum_{r: j \in r} \lambda_r \prod_{i \in r - \{j\}} (1 - \mathcal{E}_i) \tag{2.6}$$

The function $\mathcal{E}$ is Erlang's formula (2.1), and call holding times are independent and identically distributed with mean equal to one unit. The set of equations (2.5) has a unique solution that is termed the *Erlang fixed point* [36]. An approximation for the loss probability on route r is given by

$$1 - L_r \approx \prod_{j \in r} (1 - \mathcal{E}_j), \qquad r \in \mathcal{R} \tag{2.7}$$

The idea underlying the approximation is that of a *thinned* Poisson stream [32, 71] by a factor $(1 - \mathcal{E}_i)$ at each link $i \in r - \{j\}$ before being offered to link j. By assuming that these thinnings are independent both from link to link and over all routes passing through link j, then the traffic offered at link j would be Poisson at rate (2.6), the blocking probability at link j would be given by (2.5) and the loss probability on route r would satisfy (2.7) exactly. When the accuracy of this approximation was questioned, it was found that, under certain limiting conditions, the error generated from the approximation tends to zero [36]. Expression (2.7) is used frequently in practice [6, 35, 38].

2.2 Virtual Path Partitioning in ATM Networks

Communication in ATM networks is based on fixed size packets (53 bytes) called *cells*. A call is composed of a sequence of cells created by a node in the network.

the *source* node, and received by a *destination* node. This renders ATM networks connection-oriented by design. When a call is admitted into the network, a *virtual circuit* or *virtual channel* (VC) is established along the path chosen to route the connection between the given source-destination (SD) node pair.

In ATM networks, the cell loss requirement is considered the most stringent and usually dominates other performance requirements [47]. Consequently, the *virtual path* (VP) concept was introduced, and was shown to be a powerful transport mechanism for ATM networks [1, 3, 12, 22, 25, 47]. By allowing VCs to be bundled together as a single larger unit, the results are *(i)* smaller total processing requirements, *(ii)* faster processing per circuit and *(iii)* significant improvement in use of network resources. More than 90% of processing time can be saved when VCs are routed on VPs rather than processed individually [12]. The down-side, however, is an increase in loss rates since the overall resource sharing degree is reduced due to the pre-reserved capacity in VP's.

A VP is a logical connection for a node pair by means of a label in the header of an ATM cell named Virtual Path Identifier (VPI). Each VP is considered as a logical link for a certain service. VP subnetworks for different services can be built within an ATM network. The concept of virtual paths enhances the network flexibility and reliability because it decouples the logical network structure from the physical topology of the ATM network.

An uncontrolled overload of the signaling system can render inoperable most of the backbone switches that receive the highest signaling load and reduce the throughput of the transport network dramatically. In general, an ATM network supporting a Switched Virtual Circuit (SVC) connection service can overcome this problem in two ways: by blocking a portion of the incoming call setup requests at the source node, thereby preventing congestion at the downstream nodes, or by

setting up VPs between the Source-Destination pairs that contribute the bulk of the signaling load of the intermediate switches. If the first approach is followed, a call might be blocked even if there is bandwidth available in the network. Therefore, the second approach is superior, but at the expense of a reduced network throughput due to end-to-end bandwidth reservation.

At the call level, the *quality of service* (QoS) is guaranteed by bounding the blocking probability of the VC service for every Source-Destination (SD) pair in the network and the average connection setup time. The latter is achieved by modeling the signaling system as a network of queues and bounding the call arrival rates at the signaling processors. A detailed study of buffer sizing in the design phase of ATM switches under different traffic classes is given in [37].

The network manager establishes VPs within its administrative domain. This VP network is transparent to the users. It serves the purpose of carrying user calls through the public network at reduced call processing costs (Section 2.2.1). A separate routing mechanism determines how calls are to be routed through the VP network (Section 2.2.2). Each VP is assigned a number of physical links and an *effective capacity*, in terms of the maximum number of virtual circuits (VC) allowed, to assure QoS requirements. Alternatively, the *effective capacity* can be thought of as the total current bandwidth for the VP. Several VPs may be multiplexed on the same physical link. Therefore, the use of VPs has the following advantages:

- Reduction of call set-up delays. At call set-up, the routing tables of the transit nodes need not be updated. The routing procedure is also avoided at the transit nodes.

- Simpler hardware. Due to the elimination of call set-up functions, the processing load decreases. This leads to low cost node construction.

- Logic service separation on network service access.
- Simpler virtual circuit admission control.

The disadvantage is that the network throughput decreases (the total call blocking rate increases) because the degree of capacity sharing decreases.

The major challenge is to understand the trade-off between the call set-up delay and the call blocking rate. In a VP dimensioning problem the goal is to find that operating point in the network that optimally creates the balance between call set-up overload on the switches on one hand and the total call blocking rate, and consequently the total throughput on the other hand.

2.2.1 Problem Formulation of a VP Partitioning Problem

The VP distribution problem was proven to be an NP-hard [1, 3, 12] optimization problem that can be stated as follows. Given the network topology, the effective capacities of network links, the capacities of the signaling processors, and the matrix of offered load, calculate the routes and capacities of Virtual Paths in the network such that the following requirements are satisfied:

1. The sum of VP capacities on each link does not exceed its effective capacity.
2. The signaling load on each signaling processor is below a predefined upper bound.
3. The call blocking rate of each SD pair is below a predefined upper bound (also referred to as the SD blocking constraint).
4. The network revenue is maximized under the above constraints.

A small call arrival rate with high capacity demands puts pressure on the transport network, whereas for greater call arrival rates with small capacity demands, the pressure is shifted to the signaling system.

The Poisson model is adequate for modeling the call-level behaviour in broadband networks [36, 63, 71]. Therefore, it is widely used to model the call arrival process in current telephone networks. Practically, when sampling over a given period of the day characterized by intense traffic, the interarrival times between successive calls are independent from each other and from the holding times of calls. They are usually modeled by a collection of independent identically distributed random variables following an exponential distribution. The arrival process therefore belongs to a Poisson distribution [71].

The VP dimensioning problem formulation presented in the current section is based on the formulation found in [3]. Let us begin by defining the following design quantities:

- $G(V, L)$: topology of the physical network.
- W: set of SD pairs, and $\mathcal{P}$: set of VPs.
- R_w: set of routes for each SD pair w. Each route consists of an arbitrary combination of VPs and links between the source and destination nodes. $N_w = |R_w|$ is the number of routes joining the SD pair w.
- K: set of traffic classes in a multiservice network.
- Each link $l \in L$ has capacity C_l given by the *Schedulable Region* (SR). The SR is a surface in a K dimensional space that describes the allowable combinations of calls from each traffic class that can be accepted on the link and be guaranteed QoS.

- Let the state of the link $\mathbf{n} = (n^1, n^2, \ldots, n^K) \in SR_l$ where n^k denotes the number of class k calls in progress on link l, and the effective capacity of a class k call denoted by c^k, then the SR of link l is given by [63]

$$SR_l = \{\mathbf{n} | \sum_{k=1}^{K} n^k c^k \leq C_l\} \qquad l \in L \tag{2.8}$$

- An analogous definition is given for the *Contract Region* (CR) of VPs, with $C_p^l, p \in \mathcal{P}$ being the the networking capacity assigned to each VP on link l.

$$CR_l = \{\mathbf{n} | \sum_{k=1}^{K} n^k c^k \leq C_p^l\} \qquad l \in L, p \in \mathcal{P} \tag{2.9}$$

  The CR is a subregion of the SR reserved for exclusive use by a VP that can carry calls of different traffic classes. A sufficient condition to satisfy the capacity constraints for every link is:

$$\sum_{p \in \mathcal{P}} C_p^l \leq C_l \qquad l \in L$$

- λ_w^k: average Poisson call arrival rate for service class k and route w.

- μ_w^k: inverse of the average holding time for service class k and route w.

- ρ_w^k: the traffic intensity $\rho_w^k = \frac{\lambda_w^k}{\mu_w^k}$.

- μ_v: processing capacity of the node signaling processor for switch $v \in V$ in requests per unit time; and λ_v total arrival rate of call setup requests at switching node v. The signaling constraint at each switch makes sure that $\lambda_v < \mu_v$ to guarantee normal operation of every signaling processor, it thus rejects surplus of call requests violating this constraint.

- The routing policy for each set R_w with $N_w > 1$ specifies the schedule for finding available routes. Different routing policies are described in Section 2.2.2.

- β_w^k is the blocking constraint for SD pair w and traffic class k. It ensures the required QoS at the call level.
- p_w^k is the blocking probability, that is the percentage of call attempts of class k for the SD pair w that are denied service due to the unavailability of resources.
- The throughput γ_w^k of SD pair w and traffic class k is given by

$$\gamma_w^k = (1 - p_w^k)\lambda_w^k$$

- The network revenue is given by

$$Rev = \sum_{w \in W} \sum_{k=1}^{K} \gamma_w^k \alpha_w^k$$

where α_w^k is the revenue generated by accepting one call of class k on SD pair w.

The VP allocation problem can therefore be stated as follows:

Maximize *Rev*

subject to:

- the capacity constraint: $\sum_{p \in \mathcal{P}} C_p^l \leq C_l \qquad l \in L.$
- the blocking constraint: $p_w^k \leq \beta_w^k \qquad k \in \{1, 2, \ldots, K\}. \quad w \in W.$
- the signaling constraint: $\lambda_v < \mu_v \qquad v \in V.$

2.2.2 Routing Policies

Initial VP capacity allocation algorithms assume that one or more VPs are established between every SD pair, which has two major flaws [3]:

1. it is not scalable: a network with hundreds of nodes will need a very large number of VPs and consequently the VP distribution task will be overwhelming; besides, the number of VPs in ATM links is always limited by the size of the Virtual Path Identifier (VPI) header field (10 bits), and

2. there is a substantial cost associated with loss of throughput due to the rigid capacity reservation for each VP.

Therefore, a VP capacity allocation algorithm should not distribute all network capacity between VPs, but rather only a portion of it. This can be achieved by using a hybrid routing scheme, where calls first attempt to be established over a route using a direct VP, and if unsuccessful, over routes using one or more intermediate nodes and VPs or physical links in-between. Such a scheme can maintain a natural balance between the processing costs and the cost due to the reservation of resources.

For each set R_w with $N_w > 1$, the routing policy is the methodology that specifies how to select an available alternative route when there is no room to admit an incoming call on the direct VP joining the SD pair w. Several remarks should be highlighted:

- the VPs that *(i)* connect the same SD pair and *(ii)* involve the same number of physical links are jointly considered as a *macro logical link*.

- On the arrival of a VC (call), a path, which may contain multiple macro logical links, is selected from the given set of candidate paths R_w by following a certain routing policy.

- If each macro logical link on the route has sufficient spare capacity to accommodate this new VC, this new VC is admitted and routed over the selected

path: otherwise, the VC is rejected.

A thorough review of current routing policies can be found in [63]. The following routing policies are examples of existing routing schemes:

- *Sequential Routing*: this scheme sequentially examines the set of alternative routes when the direct route is full. The incoming call is admitted to the first permissible alternative route that can accommodate its bandwidth requirement on all of its links. If there is no permissible route, the call is blocked.

- *Random Routing*: an alternative route is chosen at random from the set of alternative routes. If the selected route is full, the call is blocked.

- *Dynamic Alternative Routing*: this scheme associates a two-link alternative route with each pair of switches. If at call arrival, the direct route is full and the alternative route is permissible, then the call is established in the associated alternative route; otherwise the call is blocked and a new associated alternative route is chosen for use by subsequent calls.

- *Least Loaded Routing*: this scheme keeps track of the idle capacity for each of the alternative routes. Upon call arrival, if the direct route is full, the alternative route with the most idle capacity is selected from the set of permissible routes.

2.3 Noncooperative Game Models

At the center of many disciplines, there exists a conflict situation. These disciplines include economics, applied mathematics, engineering, sociology and politics.

Game theory provides a set of analytical tools designed to help us understand the phenomena that takes place when there is a collision of interests that arises among different decision makers.

In all game theoretic models, the basic entity is a *player* or a *decision-maker*. A player may be interpreted as an individual or as a group of individuals making a decision. Such a decision should favour the player's resulting outcome, i.e., maximize a certain gain or minimize a loss. This phenomenon is best expressed by a *cost function* for each player. Formally, each player varies his own *decision variable* to optimize his cost function. Naturally, the cost function of any player depends on both his and other players' decision variables [7].

The game itself is a description of strategic interaction that includes the constraints on the actions that the players *can* take and the players' interests, but does not specify the actions that the players *do* take. For each player's action, there is a corresponding outcome. A *solution* is a systematic description of the outcomes that may emerge in a family of games.

Generally, games may be categorized in different ways. One way to classify them is as follows [60]:

- *Noncooperative or Cooperative Games*

  Once the set of players is defined, we may distinguish between two types of models: those in which the set of possible actions of *individual* players are primitives, and those in which the sets of possible joint actions of *groups* of players are primitives. The first type is referred to as "noncooperative" while the second type as "cooperative" games.

- *Strategic or Extensive Games*

  A strategic game is a model of a situation in which each player chooses his

plan of action once and for all without being informed of the plan of action chosen by any other player, and all players' decisions are made simultaneously. By contrast, in an extensive game model, each player can consider his plan of action not only at the beginning of the game but also whenever he has to make a decision. Thus an extensive model specifies the possible orders of events.

- *Games with Perfect or Imperfect Information*

  The third distinction is based on the amount of information available to each participant in the game. While in some situations, the players may be totally informed about each others' moves, in other scenarios, they may be just partially informed.

The focus will be put on noncooperative games versus the cooperative ones since, as will be explained, they constitute a more suitable modeling tool for the kind of interaction that takes place among users in a communication network. The objective of the following discussion is to highlight the nature of noncooperative games, where a discussion about the different types of equilibria and conditions of their existence is presented. [1]

One distinction that should be outlined before we proceed with our discussion about noncooperative games is the one that exists between two-player *zero-sum* and *nonzero-sum* games. Consider a game between two players, P_1 and P_2. Let u^i and J^i denote, respectively, the decision variable and cost function of the i^{th} player; $i = 1, 2$. In a zero-sum game, $\sum_{i=1}^{2} J^i(u^1, u^2) = 0$. This is the case when the amount of gain added to one player is the same as the loss incurred upon the other.

[1] It should be noted that this is not a comprehensive description of all games classifications. For a broader explanation on the topic, the reader should refer to the given references.

Therefore, the cost functions of both players has the same form but with opposite signs. Thus one player aims to minimize his loss (minimizer) while the other tries to increase his gain (maximizer). So, an equilibrium in two-players zero-sum games is called a *saddle-point* equilibrium.

Another variation of a zero-sum, is a "constant-sum" game which can be transformed to a zero-sum game through a simple translation without affecting the essential features of the game.

However, in a nonzero-sum game, the quantity $\sum_{i=1}^{N} J^i(u^1, u^2, \ldots, u^N)$ is not constant; and the definition is extended to two or more players, where N denotes the number of players. The types of equilibria that may exist in these games are presented in the subsequent discussion. As a convention, all players are cost-minimizers [7].

Finally, a *finite* game is a decision problem in which the strategy space of any player is finite. In other words, each player has a finite set of alternatives to choose from that would ultimately lead to his best possible outcome in the game (Section 2.3.1). On the other hand, in an *infinite* game at least one of the players has at his disposal an infinite number of alternatives to choose from (Section 2.3.2).

2.3.1 Finite Noncooperative Games

Nash Equilibrium in Finite Games

In the subsequent definitions for N-players noncooperative games, the following notation is used:

$\mathbf{N}$	$\{1,\ldots,N\}$ players' set; $i \in \mathbf{N}$
P_i	player i
J^i	cost function of P_i
γ^i	strategy (decision rule) of P_i
Γ^i	strategy space of P_i

Definition 2.1 *An N-tuple of strategies* $\{\gamma^{1*}, \gamma^{2*}, \ldots, \gamma^{N*}\}$ *with* $\gamma^{i*} \in \Gamma^i$, $i \in \mathbf{N}$, *is said to constitute a* noncooperative Nash equilibrium solution *for an N-player nonzero-sum finite game, if the following inequalities are satisfied for all* $\gamma^i \in \Gamma^i$, $i \in \mathbf{N}$:

$$J^{1*} \triangleq J^1(\gamma^{1*}, \gamma^{2*}, \ldots, \gamma^{N*}) \leq J^1(\gamma^1, \gamma^{2*}, \ldots, \gamma^{N*})$$

$$J^{2*} \triangleq J^2(\gamma^{1*}, \gamma^{2*}, \ldots, \gamma^{N*}) \leq J^2(\gamma^{1*}, \gamma^2, \ldots, \gamma^{N*})$$

$$\vdots$$

$$J^{N*} \triangleq J^N(\gamma^{1*}, \gamma^{2*}, \ldots, \gamma^{N*}) \leq J^N(\gamma^{1*}, \gamma^{2*}, \ldots, \gamma^N)$$

The N-tuple of quantities $\{J^{1*}, J^{2*}, \ldots, J^{N*}\}$ *is known as a* Nash equilibrium outcome *of the game.*

Existence of a Nash Equilibrium

Not every strategic game possesses a Nash equilibrium. However, if a game does possess one, then the game will have a steady state solution. Further, the existence of equilibria for a family of games allows us to study properties of these equilibria without finding them explicitly. The conditions under which the set of Nash equilibria is nonempty has been investigated extensively in [7, 60].

Let us now denote by γ_{-i} the $(N-1)$-tuple $\{\gamma^1, \gamma^2, \ldots, \gamma^{i-1}, \gamma^{i+1}, \ldots, \gamma^N\}$ of strategies. That is the set of strategies of all players except P_i.

Definition 2.2 *In a N-player finite game, we call the set-valued function R^i the* optimal response function *of player i. It is defined as follows:*

$$R^i(\gamma_{-i}) = \{\varepsilon \in \Gamma^i : J^i(\varepsilon, \gamma_{-i}) \leq J^i(\gamma^i, \gamma_{-i}) \ \forall \gamma^i \in \Gamma^i\}$$

To show that a game has a Nash equilibrium, it suffices to show that there is a profile $\gamma^* = \{\gamma^{1*}, \gamma^{2*}, \ldots, \gamma^{N*}\}$ of strategies such that [7]

$$\gamma^{i*} \in R^i(\gamma_{-i*}) \qquad \forall i \in \mathbf{N} \tag{2.10}$$

Equation (2.10) states that the intersection of the $\mathbf{N}$ R^i sets $\forall i \in \mathbf{N}$ constitutes the set of Nash equilibrium points. This notion is graphically illustrated in the case of infinite noncooperative games in Section 2.3.2.

Fixed point theorems [68] provide conditions on R under which there exists a value of γ^* for which $\gamma^* \in R(\gamma^*)$, i.e. the intersection of the R^is is a nonempty set.

Theorem 2.1 *Let Γ be a compact convex subset of $\mathbf{R}^n$ and let $R : \Gamma \to \Gamma$ be a set-valued function for which [31, 68]*

- *for each $\gamma \in \Gamma$, the function $R(\gamma)$ is nonempty, convex and assigns a closed and convex subset of Γ.*
- *the function $R(\gamma)$ is upper-semicontinuous for all $\gamma \in \Gamma$.*

then there exists at least one $\gamma^* \in \Gamma$ such that $\gamma^* \in R(\gamma^*)$.

Stackelberg Equilibrium in Finite Games

The Nash equilibrium solution concept presented above provides a reasonable noncooperative equilibrium solution for nonzero-sum games when the roles of the

players are symmetric and there is no single player dominating the decision process. However, there are yet other types of noncooperative decision problems wherein one of the players (or more) has the ability to enforce his strategy on the other players, and for such decision problems one has to introduce a hierarchical equilibrium solution concept. The player who holds the powerful position in such a decision problem is called the *leader* and the other players who react rationally to the leader's decision (strategy) are called the *followers*. In games with multi-levels of hierarchy in decision making, the equilibrium solution is called a "*Stackelberg*" equilibrium.

Definition 2.3 *In a two-player finite game with P_1 acting as the leader and P_2 as the follower, the optimal response of P_2 for any strategy γ^1 of the leader is $R^2(\gamma^1)$ given by Definition 2.2. $\gamma^{1*} \in \Gamma^1$ is called a* Stackelberg equilibrium strategy *for the leader, if*

$$J^{1*} \triangleq \max_{\gamma^2 \in R^2(\gamma^{1*})} J^1(\gamma^{1*}, \gamma^2) = \min_{\gamma^1 \in \Gamma^1} \max_{\gamma^2 \in R^2(\gamma^1)} J^1(\gamma^1, \gamma^2)$$

The quantity J^{1*} is the Stackelberg cost *of the leader, and any element $\gamma^{2*} \in R^2(\gamma^{1*})$ is an* optimal strategy *for the follower that is* in equilibrium *with γ^{1*}. The pair $\{\gamma^{1*}, \gamma^{2*}\}$ is a* Stackelberg solution *for the game.*

Further, if $R^2(\gamma^1)$ is a singleton for each $\gamma^1 \in \Gamma^1$, then there exists a mapping $T^2 : \Gamma^1 \to \Gamma^2$ such that $\gamma^2 \in R^2(\gamma^1)$ implies $\gamma^2 = T^2\gamma^1$. This corresponds to the case in which the optimal response of the follower (given by T^2) is unique for every strategy of the leader, thus leading to a more simplified version for the cost

$$J^{1*} \triangleq J^1(\gamma^{1*}, T^2\gamma^{1*}) = \min_{\gamma^1 \in \Gamma^1} J^1(\gamma^1, T^2\gamma^1)$$

Here J^{1*} is no longer only a secured equilibrium cost level for the leader, but it is the cost level that is actually attained.

In an attempt to minimize his cost, the leader P_1 accounts for the worst case scenario in response to a given strategy γ^1, by assessing his maximum cost for each strategy $\gamma^2 \in R^2(\gamma^1)$ and selecting the strategy γ^{1*} that results in the least maximum possible. This is what the "min max" term in Definition 2.3 means.

For a given two-player finite game, let J^{1*} again denote the Stackelberg cost of the leader (P_1), and J^1_N denote any Nash equilibrium cost for the same player. J^{1*} is not necessarily lower than $J^1{}_N$ when the optimal response of the follower is not unique. However, if $R^2(\gamma^1)$ is singleton for each $\gamma^1 \in \Gamma^1$, then

$$J^{1*} \leq J^1{}_N$$

In other words, the leader never does worse in a "Stackelberg game" than in a "Nash game" [7].

Extension of the above discussion to N-player games could be done in a number of ways. For example, for $N = 3$, there are three different modes of play:

- There are two levels of hierarchy in decision making: one leader and two followers. The followers react to the leader's announced strategy by playing according to a specific equilibrium concept among themselves, Nash for instance.

- Two leaders and one follower. There are still two levels of hierarchy in decision making. Now the leaders play according to a specific equilibrium concept among themselves, by taking into account possible optimal responses of the follower.

- There are three levels of hierarchy. First P_1 announces his strategy, then P_2 determines his strategy by also taking into account possible responses of P_3

and enforces this strategy on him, and finally P_3 optimizes his cost function in view of the announced strategies of P_1 and P_2.

These different patterns of "relative leadership" can conceptually be generalized to N-player games with $N > 3$.

2.3.2 Infinite Nonzero-sum Games

The following discussion is the "continuous" counterpart of the one presented in Section 2.3.1. The notion of a reaction set within the context of finite games is now replaced by a curve, or a family of curves in infinite games. Depending on the structure of the action sets and the cost functions of the players, the properties of these curves, such as continuity and differentiability, are determined. The previous notation still holds, except the γ/Γ notation has changed to:

u^i strategy (decision rule) of P_i

U^i strategy space of P_i

to reflect the change from the discrete domain to the continuous domain of the strategy space for each player.

Definition 2.4 *In an N-player nonzero-sum game, let the minimum of* $J^1(u^1, \ldots, u^N)$ *with respect to* $u^1 \in U^1$ *be attained for each* $u_{-1} \in U_{-1}$, *where* $u_{-1} \triangleq \{u^2, \ldots, u^N\}$ *and* $U_{-1} \triangleq U^2 \times \cdots \times U^N$. *Then, the set* $R^1(u_{-1}) \subset U^1$ *defined by*

$$R^1(u_{-1}) = \{\varepsilon \in U^1 : J^1(\varepsilon, u_{-1}) \leq J^1(u^1, u_{-1}), \quad \forall u^1 \in U^1\}$$

is called the optimal response *or* rational reaction set *of* P_1. *If* $R^1(u_{-1})$ *is a singleton for every* $u_{-1} \in U_{-1}$, *then it is called the* reaction curve *or* reaction function *of* P_1, *and is denoted by* $l_1(u_{-1})$. *Similar definitions are given for each player.*

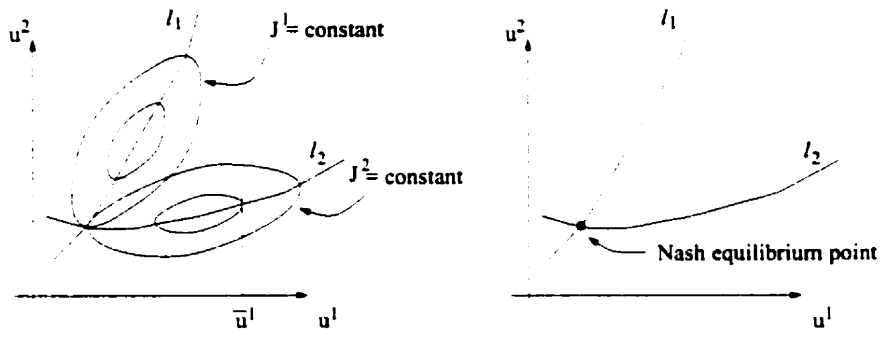


Figure 2.1: Iso-cost curves and the corresponding reaction curves for a two-player infinite game.

Figure 2.1 illustrates the role of reaction curves in the derivation of Nash equilibria. For each player P_i, $i = 1, 2, \ldots$, *iso-cost* curves are drawn corresponding to his cost function. The strategy spaces are uni-dimensional, i.e., $U^1 = U^2 = \mathbf{R}$. For a fixed value of u^1, say $u^1 = \bar{u}^1$, P_2 tries to minimize his cost along the line $u^1 = \bar{u}^1$. If the optimal solution to this problem is unique, this solution is graphically the point where the line $u^1 = \bar{u}^1$ is tangent to iso-cost curve $J^2 = \text{constant}$. The collection of these points for different values of u^1 forms the reaction curve l_2 of P_2. The reaction curve of P_1 is constructed in a similar way.

By definition, the Nash solution must lie on both reaction curves. Therefore, if these curves have only one intersection point, as in Figure 2.1, the Nash solution exists and is unique. There can be more than one intersection point thus creating a set of Nash equilibria. In cases where the reaction curves are parallel straight lines, the set of Nash equilibria is empty. By contrast, when the reaction curves partly coincide there exists a continuum of Nash solutions.

Nash Equilibria in Infinite Games

In light of the previous discussion, the existence of Nash equilibria in N-player

infinite games can be established by proving the existence of well-defined reaction functions with a common point of intersection.

Theorem 2.2 *For each $i \in \mathbf{N}$, let U^i be a closed, bounded and convex subset of a finite-dimensional Euclidean space, and the cost function $J^i : U^1 \times \cdots \times U^N \to \mathbf{R}$ be jointly continuous in all its arguments and strictly convex in u^i for every $u^j \in U^j, j \in \mathbf{N}, j \neq i$. Then, the associated N-player nonzero-sum game admits a Nash equilibrium.*

The strict convexity guarantees a unique mapping $l_i : U^{-i} \to U^i$ for each player P_i, such that $u^i = l_i(u^{-i})$ (recall Definition 2.4) uniquely minimizes $J^i(u^i, u^{-i})$ for any given $(N-1)$-tuple u^{-i}, $i \in \mathbf{N}$. In addition, the reaction curves $l_i, i \in \mathbf{N}$ are continuous in their arguments due to the compactness of the strategy spaces. Using vector notation, these relations can be written in a compact form as $u = L(u)$, where $u = (u^1, \ldots, u^N) \in U \triangleq U^1 \times \cdots \times U^N$, and $L = (l_1, \ldots, l_N)$. From the fixed point theorem (Theorem 2.1) there exists a $u^* \in U$ such that $u^* \in L(u^*)$, that is, u^* is a fixed point of L. The individual components of u^* constitute a Nash equilibrium solution.

Stackelberg Equilibria in Infinite Games

The discussion about Stackelberg equilibria is analogous to its counterpart in finite games, except that the strategy spaces of the players are taken over finite-dimensional spaces in infinite games, with a continuum of alternatives for a player to choose from. Definition 2.5 gives the definition of the Stackelberg equilibrium in infinite game and it is similar to Definition 2.3 for the finite games case. Further, each mapping function T^i is now substituted by player's i reaction curve l_i.

Definition 2.5 *In a two-player game, with P_1 as the leader, a strategy $u^{1*} \in U^1$ is called a* Stackelberg equilibrium strategy *for the leader if*

$$J^{1*} \triangleq \sup_{u^2 \in R^2(u^{1*})} J^1(u^{1*}, u^2) \leq \sup_{u^2 \in R^2(u^1)} J^1(u^1, u^2)$$

for all $u^1 \in U^1$. Further, if $R^2(u^1)$ is singleton for each $u^1 \in U^1$, which means that it is described completely by a reaction curve $l_2 : U^1 \to U^2$, then the above definition becomes

$$J^{1*} \triangleq J^1(u^{1*}, l_2(u^{1*})) \leq J^1(u^1, l_2(u^1))$$

for all $u^1 \in U^1$.

It should be noted that the Stackelberg equilibrium solution exists under a set of sufficiency conditions which are much weaker than those required for existence of Nash equilibria stated in Theorem 2.2. For the sake of a simpler representation, we illustrate the Stackelberg equilibrium for a two-player game depicted in Figure 2.2. With P_1 as the leader, the Stackelberg solution will be situated on l_2, at the point

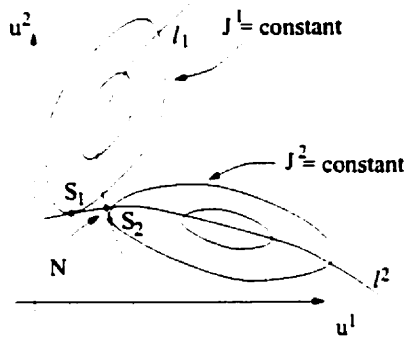


Figure 2.2: Nash and Stackelberg equilibria

where it is tangent to the appropriate iso-cost curve of J^1 (point S_1). The coordinates of S_1 (respectively, S_2) correspond to the Stackelberg solution of the game when P_1 (respectively, P_2) is the leader.

The point of intersection of the reaction curves l_1 and l_2 (point N) denotes the unique Nash equilibrium solution of this game. It should be noted that the Nash costs of both players are higher (worse) than their corresponding equilibrium costs in their Stackelberg game. This is only true due to the fact that their optimal response sets ($R^1(u^2)$ and $R^2(u^1)$) are singleton. Therefore, in nonzero-sum games with unique follower responses, the leader never prefers to play the "Nash game" instead of the "Stackelberg game", whereas the follower could prefer to play the "Nash game", if given the option.

Chapter 3

Hierarchical Network Management

Recently, it has been recognized that system-wide, single (centered) administration is an impractical paradigm for the control of giant broadband networks. Moreover, the practical difficulties of coordinating among managers in large-scale networks often require that each manager would effectively attempt to optimize its own performance. According to this formulation, the network is a common resource shared and competed over by "selfish" users. This is a typical scenario of a non-cooperative game.

Chapter 2 presented a review of different approaches to solve the VP dimensioning problem where the dynamics of the system were modeled based on different assumptions predicting the connections behaviour, using a classical way to tackle the VP problem in Section 2.2. Whereas the brief game theory background, given in Section 2.3, is useful in understanding the foundational work of some of the

recent studies that use game theory to model the user [1] competitive interaction within the network, including the current work. We believe that the game model captures better the dynamic behaviour of users than the former approaches because it is based on the plausible assumption that users act in a greedy manner trying to minimize their own cost functions. This assumption is validated and proved as will be displayed by the simulation results in Chapter 5. This is the approach adopted in the current work.

In the present chapter, we describe a hierarchical technique for Virtual Paths dimensioning in large-scale networks. The network is organized in a hierarchical structure described in Section 3.1. In Section 3.2, the global network management and the overall traffic control are explained by describing the communication between the different management modules or controllers distributed across the network. Furthermore, the optimization of the VP configuration problem that takes place locally in each subnetwork is defined.

3.1 Hierarchical Network Structure

When the population of users is large, propagating information for each link throughout the network quickly becomes unmanageable as the size of the network increases. One possible approach for complexity reduction, embodied in the ATM Forum PNNI standard [17], is to introduce a hierarchical process that progressively "aggregates" state information as networks get more and more remote.

In general, the accuracy of the information decreases as a function of the amount of aggregation that has been performed. For instance, in the context of the PNNI

[1] The terms "user" and "connection" are used interchangeably.

hierarchical model, the metrics and topology information on higher level nodes and links is typically very approximate. This is due to the fact that such "nodes" and "links" actually represent an increasingly larger number of underlying nodes, links and even networks.

Nevertheless, the emerging standards for ATM networks favour such an approach [70]. We can briefly cite some of the advantages of a hierarchical approach:

- it reduces the amount of exchanged information.
- it makes addressing feasible in a large-scale network, as demonstrated by the network addressing of virtual paths due to the VPI field limitations (10 bits).
- it permits the use of different routing schemes at different levels of the hierarchy.
- it provides more flexibility in terms of increasing or down-sizing the network, i.e. it is scalable.

Figure 3.1 illustrates the idea of conceptually grouping nodes at a lower level into a single node in the following higher level in the hierarchy.

Groups of switches are organized into *peer groups* (clouds), and peer group leaders are chosen to coordinate the representation of each group's state. These collections of switches then form peer groups at the next level of the hierarchy and so on. The logical view of the network from a given peer group's perspective consists of *complete* information for all links within the peer group but only *aggregate* information for links between peer groups and in other peer groups [24, 57].

In a hierarchical structure, the network or set of networks, across which connections need to be routed, is represented by a graph $G(V, L)$, where V is the set

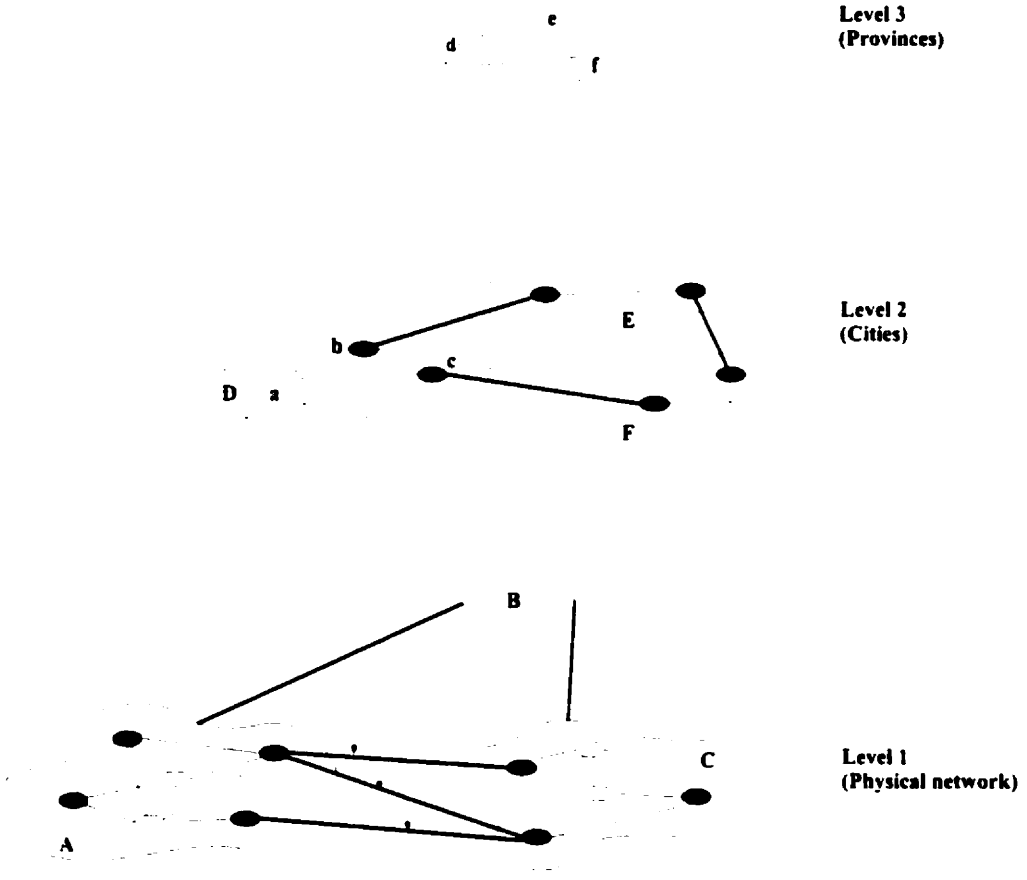


Figure 3.1: Hierarchical organization in a large-scale network.

of *layer (1)* switches (Figure 3.1), and L is the set of links interconnecting them. $G(V, L)$ represents the *actual (physical) network*. Layer (1) switches are clustered to form layer (2) nodes, which are clustered into layer (3) nodes, and so on, up to the last K^{th} layer. Nodes in layer (k), that are grouped together into the same layer $(k+1)$ node, are said to belong to the same *peer group* [17]. They are also the *children* of the layer $(k+1)$ node, which is, likewise, called their *parent* node. The structure can be described as follows.

- Consider the network with V switching nodes interconnected by L links, denote the first layer by $G(V, L)$ in Figure 3.1. We base our model on the assumption that the network is already segmented, i.e. the partitioning of layer (1) nodes into peer groups or clouds is already given. This partitioning can follow geographical or national divisions. For instance, in Figure 3.1, the layer (1) clouds represent different cities, within different provinces (layer (2)), that in their turn are part of a given country (layer (3)).

- The link states in a peer group are aggregated to form the state of the corresponding logical node at the higher hierarchical level. Typically, only a subset of the nodes in a cloud are *border nodes* (shaded nodes in Figure 3.1) through which transit paths may enter or leave the cloud. A link state aggregation method is required to produce an aggregated topology with $V' \leq V$ border nodes, and $L' \ll L$ logical links, such that enough information may be derived for efficient admission and routing of connections. Section 3.2.2 outlines one such approach for information aggregation throughout the network layers.

- Links interconnecting clouds (for example, the set of links joining the cloud pair A and C) are aggregated into a single link joining node pair (a) and (c) in their parent logical node (D). The pattern of information flow between

adjacent layers is explained in Section 3.2.1.

3.2 VP Partitioning and Routing Process

The network topology and the traffic demands that need to be routed constitute the input to the main module, the *VP Partitioning/Routing Processor*. The global network management paradigm is explained in Section 3.2.1. Whereas, the mathematical model details and the problem formulation for the individual network are described in Section 3.2.3. The explanation of the actual implementation of this module is deferred to Section 4.2.2.

3.2.1 Global Network Management

Given a large-scale network organized in a hierarchical structure as described in the previous section, the issue to address now is, from a functional point of view, how do network managers coordinate the task of optimizing the VP configuration for the whole network?

For the actual VP configuration problem for a set of traffic demands using the global network resources, we introduce the following distributed algorithm:

- Each node, in the second hierarchical level and up, simply represents a *local* network controller (or manager) responsible for the VP architecture of its children.
- At each hierarchical level, all local network managers operate in parallel. However, each layer (k) network manager sequentially follows the operation of his layer ($k+1$) parent.

- Given the traffic demands through the global network, each bandwidth request i is defined by the tuple $\{s_i, d_i, c_i, k_i\}$ where s_i and d_i denote the source and destination nodes respectively, while c_i denotes the amount of bandwidth required; and finally, k_i indicates the demand level.

- the *level* of a demand is specified as follows. The layer (1) node s_i submits a bandwidth reservation request for a destination d_i to the corresponding layer (1) network manager. The latter verifies whether d_i is within its administrative domain. In other words if both s_i and d_i are within the same peer group, then the demand i will be served by the layer (1) network manager. If, on the other hand, d_i is not within the same peer group as s_i, then the request is passed to the parent node manager and the same verification is repeated at the next level, and so on, until it falls in one of the K levels.

- Starting at the single layer (K) node, the VP configuration for all layer (K) demands is found according to the procedure that will be described in Section 3.2.3. As an illustration, let us consider the example shown in Figure 3.2. Suppose that a level-K demand (i) is described by $\{f, j, 5, K\}$. Suppose also that the VP allocation procedure run by the layer (K) manager resulted in the routing shown in Figure 3.2.(b); that is 2 units are reserved on path (f-j), 2 units on path (f-i-j), and the remaining unit is to be routed through node (B). According to this scenario, the layer ($K-1$) manager in node (B) inherits a $\{g, i, 1, K\}$ demand. From the individual connection perspective, its demand is split among different routes who belong to different hierarchical levels, where in each level, the VP configuration for the transferred connection demand is optimized with respect to the other connections competing at that particular level. Section 3.2.3 gives the details of the VP configuration

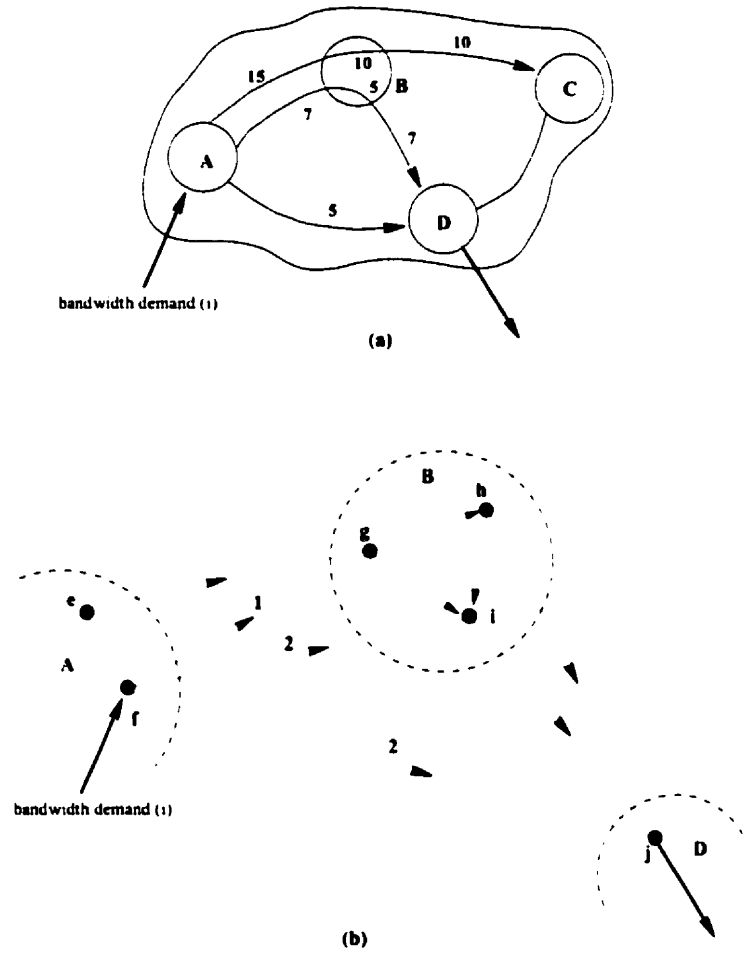


Figure 3.2: VP allocation and routing for a level-K demand.

problem within a given subnetwork or peer group.

- Along with the demand definition, the layer (K) manager passes also the blocking probabilities on links (f-j) and (i-j) to node (B)'s controller. This type of information is needed by the latter so that the overall blocking that demand (i) experiences can be calculated finally by the layer (1) manager when all the physical links on its path would be identified. Figure 3.3 illustrates the types of information that is exchanged between the network layers.

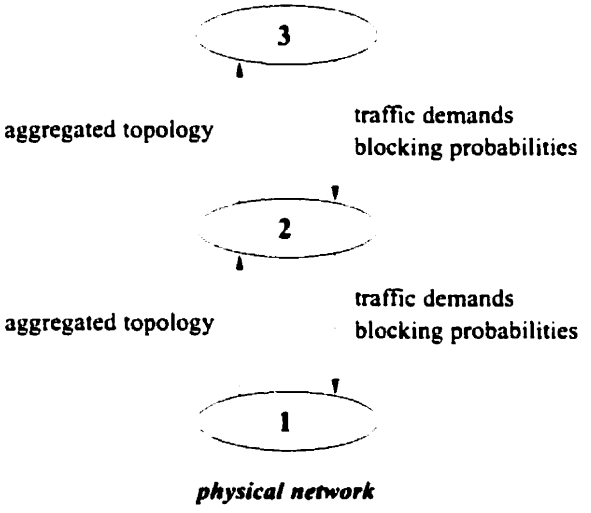


Figure 3.3: Information exchanged between network layers.

- All network managers at the ($K-1$) layer start their operation simultaneously once the layer (K) network manager has finished its own VP allocation task. The VP configuration process continues in the same manner until it is performed by all layer (1) managers.

Due to the availability of resources, parallel or distributed computation is not feasible. Instead, a sequential operation takes place in which all managers in a given

layer optimize their VP partitioning sequentially. Naturally, this scheme is more time-consuming than its parallel counterpart. Nevertheless, the overall run-time for the network does not prohibit it from being carried-out in a more dynamic set-up. However, the original objective was to construct an off-line or static partitioning of the Virtual Paths. The following chapter displays some run-time values with an estimate of their parallel computation run-time values.

The technique outlined above is a form of a multi-level optimization scenario. A comprehensive survey about other problems that made use of the multi-level optimization approach can be found in [55].

3.2.2 Topology Encoding in Subnetworks

A link state captures appropriate information pertaining to a link such that it can be used for efficient path selection and allocation of network resources for establishing connections between end-users of the communication network. The link state information must be exchanged and maintained up-to-date among all nodes in the network for path selection purposes.

A link is characterized by a vector of link states such as bandwidth and delay. There may be one aggregated topology for each link state. The aggregated topologies need not be the same since only the link state information derived from them matters. A link state may be additive or non-additive.

- An additive link state is one whose value associated with each link along a path is added up to determine if the path is acceptable for establishing a connection. The delay is an additive link state.

- A non-additive link state is one whose value associated with a link indepen-

dently determines if the link is acceptable to be part of a path for establishing a connection. The bandwidth is an example of a non-additive link state. The bandwidth of a given path is determined by the minimal bandwidth of the links along that path.

Link state aggregation is needed in large networks for two reasons:

1. the amount of link state information must be reduced to avoid excessive complexity in link bandwidth updates.
2. the internal topology of a network may have to be hidden for security reasons.

There exist different alternatives in the literature for link state aggregation [46]:

- The *Symmetric-Point* approach where the entire peer group is collapsed into one point. The all-in-one node parameter (usually the "worst case" parameter) is advertised. This approach greatly reduces the amount of information, but unfortunately, it does not capture any multiple connectivity in the original peer group.
- The *Full-Mesh* approach uses a logical direct link between each pair of border nodes to construct the aggregated topology. The collective link state information is adequate for efficient routing and network resource allocation. Unfortunately the increase in the amount of information to be advertised is proportional to the square of the number of border nodes, of $O(V'^2)$ complexity. Despite the fact that the full-mesh approach offers greater flexibility, it may as well result in a considerable redundancy when the topology of the original peer group is relatively symmetrical from the perspective of the border nodes.

- The *Star* approach is a compromise between the two extremes described above. It is considered an extension of the symmetrical-point approach where the "center" of the peer group is represented by a "pseudo-node" that is connected to the border nodes by logical links whose link states are advertised. The complexity of this approach is $O(V')$ that is, of a linear order in the number of border nodes. Like the symmetric-point approach, the star approach does not adequately capture any multiple connectivity in the peer group.

In the present system, the full-mesh of link states is constructed based on the amount of bandwidth that each link can accommodate. Links delays were not considered on the basis that they add a constant term in the definition of the quality of service (QoS) of connections [21]. To overcome the problem of the advertised information size in this type of representation, and for the sake of simplification, we advertise only the average value of the constructed logical links states.

In Figure 3.4, the logical link connecting any pair of border nodes is a representation of the maximum-flow that can take place between the given nodes in a given direction. The minimum of the two values of maximum-flow for each direction is selected to represent the bandwidth that the logical link can accommodate.

3.2.3 Capacity Allocation in Local Subnetworks

In the previous section, we described the organization of a hierarchical network and the types of interactions taking place between the different layers of the network. In the current section, we focus on the core problem of capacity allocation during the VP configuration phase, and explain the factors that shape the optimization problem at the center of each subnetwork. In other words, our problem can be

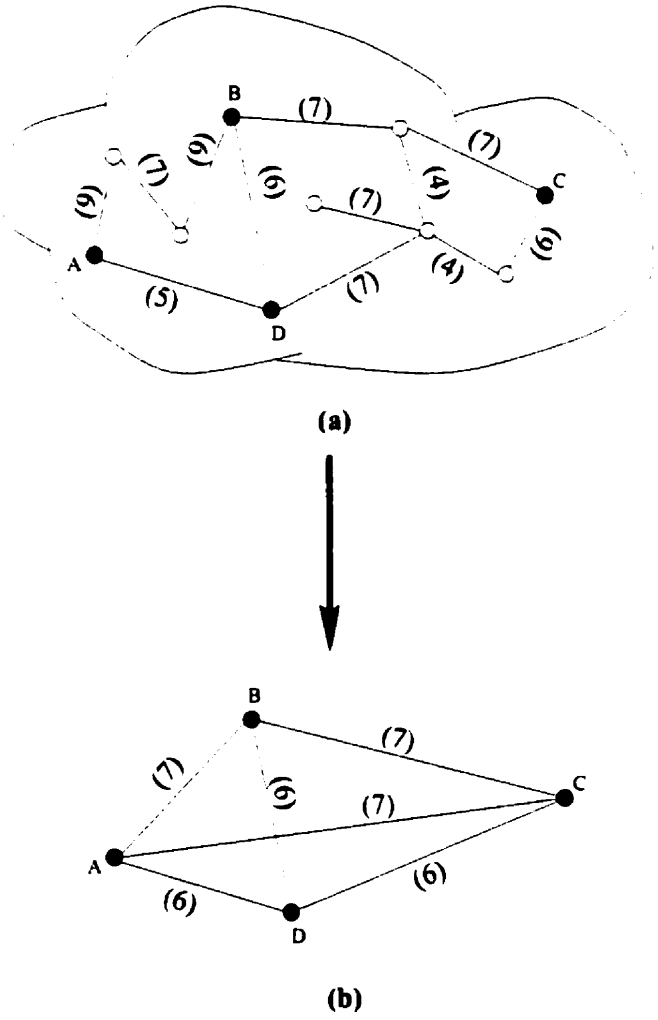


Figure 3.4: Topology encoding by Full-mesh construction.

stated as follows: given the topology and the traffic demands in a local peer group, what is the VP layout that would optimize its performance measures?

Based on the background provided in Chapter 2, we model the traffic demands competing for the resources in the local subnetwork as a set of "greedy" players, each with its own goal to optimize a certain individual objective in a noncooperative game context. However, before we proceed to the explanation of the current model, some observations are in order as to highlight the differences between the model at hand and the conventional way in which the game theory is used in the telecommunication networks.

Conventional Game Models for VP Partitioning

In the literature, the general approach to tackle the VP partitioning problem could have various forms. A typical form is to assume that all connections in a given network act as competitive players in a noncooperative game, then proceed to prove the existence and/or uniqueness of the Nash equilibrium (equilibria) of the underlying game, such as the work found in [39, 40, 42, 45].

Another way to deal with the VP partitioning problem is to introduce an extra player to the set of noncooperative players or users and assign a high priority to that particular player, namely the network manager or provider. In this scenario, the network manager becomes the leader in the resulting Stackelberg game in which the competing users become the followers who set their strategies once the leader has chosen and fixed his own [41, 43, 44].

More precisely, in this Stackelberg game context, the leader would select his strategy by shaping the individual payoff or cost functions by which the users are taxed in such a way that would achieve the best possible overall outcome from the *network*'s point of view. In their turn, the users respond to the leader's strategy by

trying to minimize their cost based on the assigned payoff functions by shaping their traffic characteristics such as the grade of service or the amount of bandwidth they request. Ultimately, when the self-optimizing users reach their final decisions, i.e. the Nash equilibrium point, the result would coincide with the original objective of the leader, who could drive the equilibrium point to the desired network operation point.

The formulation of such problems would generally follow a notation of the form:

- The set of $N + 1$ users with the network manager being "user 0".
- r: total followers demand; $R = r + r^0$: total demand.
- $N^0(\mathbf{f}^0)$: Nash equilibrium of the users induced by strategy $\mathbf{f}^0$ of the manager.
- $(f_1^*, f_2^*, \ldots, f_L^*)$: link flow configuration that minimizes the overall cost:

$$J(\mathbf{f}) = \sum_{i=0}^{N} J^i(\mathbf{f})$$

- A *maximally efficient* strategy for the manager or the leader is a strategy $\mathbf{f}^0$ such that, if $\mathbf{f} = N^0(\mathbf{f}^0)$, then for every link l:

$$f_l^0 + \sum_{i=1}^{N} f_l^i = f_l^*$$

  i.e. a strategy that enforces the network optimum.

It is essential to note that the work developed in the referenced research was mainly investigating the modeling characteristics for a single pair of nodes joined by L parallel links. It is, therefore, mandatory to point out that, unlike the present work, a closed-form solution to the optimal network operating point was obtainable. In the present work, since large-scale networks are used to verify the validity of

the current model, the problem is intractable in nature due to the combinatorial size of the different source-destination pairs. Therefore, simulation of the current technique is the main vehicle by which this work is analyzed.

Formulation of the Present Model for VP Partitioning

Let us consider a local subnetwork $G(V, L)$, where V is a finite set of nodes and $L \subset V \times V$ is a set of directed links, such that each link $l \in L$ has a bandwidth B^l. Without loss of generality, we assume that at most one link exists between each pair of nodes, in each direction. For each link $l \in L$, let $S(l)$ denote the node at the starting point of l and $D(l)$ denote the node at the ending point.

Furthermore, let $\mathcal{I}_k = \{1, 2, \ldots, N_k\}$, $k \in \{1, 2, \ldots, K\}$ denote the set of layer (k) demands competing for the bandwidth within the subnetwork G. In a

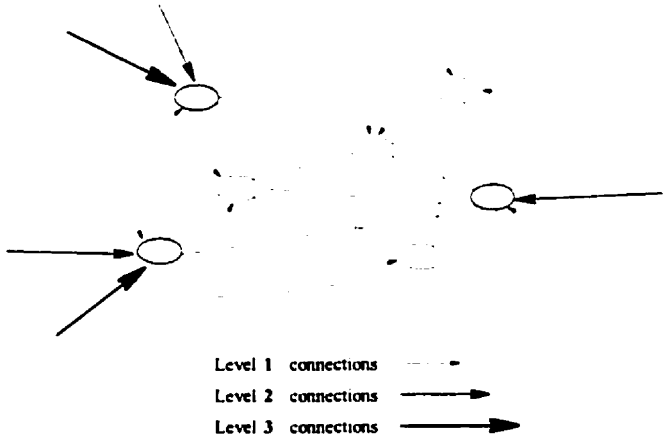


Figure 3.5: Example of traffic demands in a layer (1) subnetwork, with $K = 3$.

local layer (k) subnetwork, the network manager deals with $k, k+1, \ldots, K$ demand types. Recall that a level-k connection means that it is administered by a layer (k) network manager. Figure 3.5 depicts a plausible pattern of traffic demands found in a layer (1) subnetwork.

Let us first define the following variables:

c_i^l is the amount of capacity reserved by player i on link l.

C^l is the total capacity reserved by all connections on link l. $C^l = \sum_{k_l=k}^{K} \sum_{i \in \mathcal{I}_{k_l}} c_i^l$.

c_i is the total capacity reserved by the i^{th} connection on all simple paths (routes) joining the node pair (s_i, d_i). $c_i = \sum_{l:S(l)=s_i} c_i^l$.

In Figure 3.5, the set of demands constitute the set of players in a noncooperative Stackelberg game:

- The strategy space of player i is the capacity vector $C_i = \{c_i^1, c_i^2, \ldots, c_i^L\}$.
- The set of players $\mathcal{I}_3$ are the leaders of the game, followed by the set $\mathcal{I}_2$ players, and finally by the followers in the $\mathcal{I}_1$ set. The reason for this ordering comes from the fact that the higher the index of the call, the more revenue it produces. Therefore, the network manager, at any level, trying to maximize the network's revenue, assigns different priorities to the calls according to their index.
- A Nash game occurs between the first set of players. The resulting Nash Equilibrium Point (NEP) is denoted by $\mathbf{C}^3$. The links capacities are updated accordingly.
- The equilibrium point strategy $\mathbf{C}^3$ becomes the leader's strategy for the followers in the subsequent $\mathcal{I}_2$ set. A second NEP point is reached and is denoted by $\mathbf{C}^2$. Note that $\mathbf{C}^2 \in R(\mathbf{C}^3)$ (recall Definition 2.5). Once again, the links capacities are updated.
- Finally, the NEP of the $\mathcal{I}_1$ players is $\mathbf{C}^1 \in R(\mathbf{C}^2, \mathbf{C}^3)$.

Figure 3.6 outlines the algorithm of VP dimensioning executed by each local network manager. In the previous discussion, we assumed that there exists at least

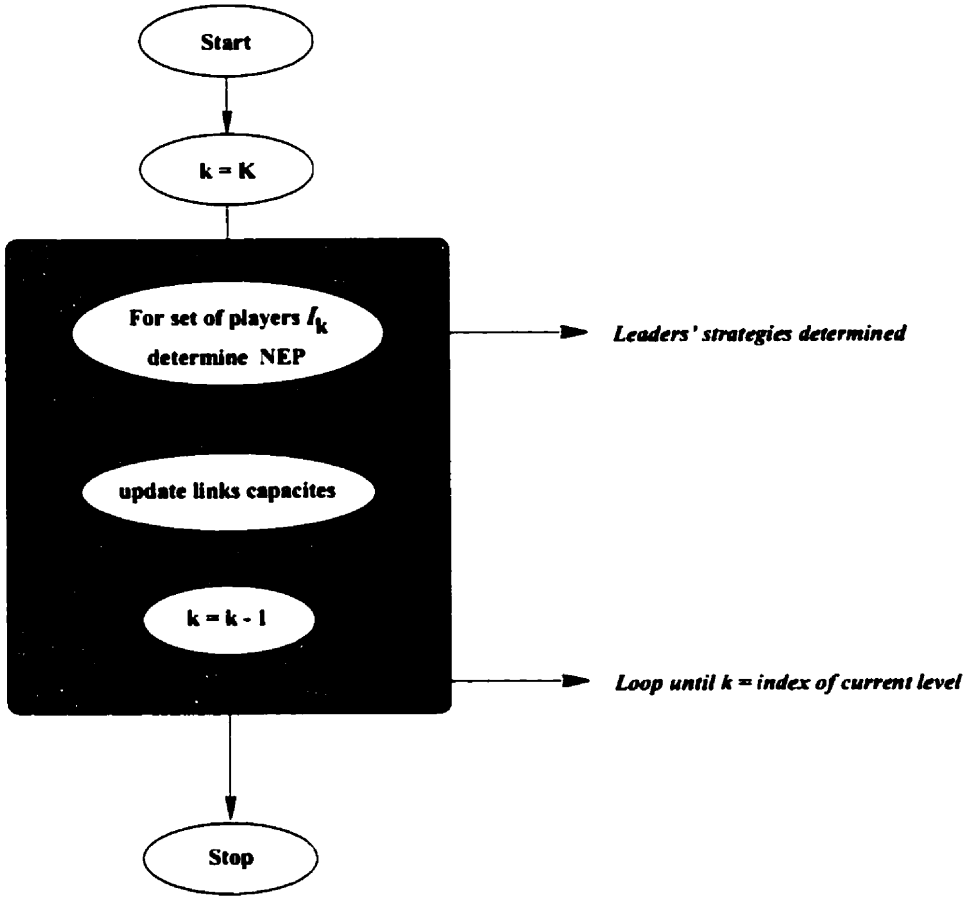


Figure 3.6: Functional description of a local network manager.

one NEP at each level. In the sequel, we present the validity of this assumption.

Each player i in the set $\mathcal{I}_k = \{1, 2, \ldots, N_k\}$ tries to reserve capacity c_i^l between source node s_i and destination node d_i on link l subject to the following:

- $c_i^l \geq 0$,
- $c_i^l \leq B^l$ where B^l is the capacity of link l.

- $\forall u \neq s, d$: $\sum_{l:S(l)=u} c_i^l = \sum_{l:D(l)=u} c_i^l$, i.e., the reserved capacity outgoing from a tandem node must be equal to that ingoing to that node, since an excess in capacity in any direction is of no benefit.

From the restrictions above, the capacity reservation problem is a network flow problem [23], with the maximum flow being the solution to the problem of the total capacity that player i can reserve in G.

In the game, each player tries to optimize its objective function. In our case, the objective function for player i is described by:

$$J_i(\mathbf{C}) = \sum_{l \in L} F_i^l(c_i^l, C^l) + G_i(c_i) \tag{3.1}$$

where the F_i^l function accounts for the availability of resources on link l perceived by the i^{th} connection, whereas the function G_i accounts for the effect that the amount of reserved capacity has on the performance of that particular connection. The definition of these two functions is as follows:

$$F_i^l(c_i^l, C^l) = c_i^l \cdot \frac{\alpha_i}{1 - C^l/B^l} \tag{3.2}$$

and

$$G_i(c_i) = \begin{cases} \frac{1}{\kappa_i - \mathcal{E}(\rho_i, c_i)}, & \text{if } \mathcal{E}(\rho_i, c_i) < \kappa_i \\ \infty, & \text{otherwise.} \end{cases} \tag{3.3}$$

where

$\mathcal{E}(\rho_i, c_i)$ is the piece-wise continuous (interpolated) Erlang loss formula described by Equation(2.1).

$\kappa_i \leq 1$ is an upper bound on connection i's call blocking probability as determined by its QoS requirements.

In the above formulation, we can see that the F function has the following properties [41, 45]:

1. $F_i^l(c_i^l, C^l)$ monotonically increases in each of its two arguments.
2. $F_i^l(c_i^l, C^l)$ is continuously differentiable with respect to c_i^l.
3. $F_i^l(c_i^l, C^l)$ is convex in c_i^l.
4. $\partial F_i^l(c_i^l, C^l)/\partial c_i^l$ is nondecreasing with respect to C^l.
5. $\lim_{c_i^l \to B^l} F_i^l(c_i^l, C^l) = \infty$ which prevents any connection from exhausting the link resources on its own.

A connection can be routed on any path (or combination of paths) between the source and destination on which the amount of reserved and unused capacity can accommodate its demand size. This means that the loss process depends only on the total amount of capacity reserved by that connection on all paths, and not on the precise distribution of that capacity among the paths. Consequently, the cost function G_i takes as its argument the total capacity c_i. Due to the nodal conservation of capacities, the total capacity c_i is equal to the sum of capacities reserved on links outgoing from the source node s_i (equivalently, the sum of capacities reserved on links ingoing the destination node d_i). Thus, the G_i function has the following properties [54]:

1. $G_i(c_i)$ is continuously differentiable.
2. $G_i(c_i)$ is strictly decreasing.
3. $G_i(c_i)$ is convex.

4. $\lim_{c_i \to 0} G_i(c_i) = \infty$.

This last property indicates that each connection needs *some* positive amount of capacity. By inspection, we can see that the F function guards against the violation of the link capacity constraint, while the G function is responsible for the blocking constraint discussed in Chapter 2.

From the above definitions, we can see that the cost function of each connection depends on the strategies of all the other connections and we are dealing with a noncooperative game situation. Recall that the set of players is composed of the connections belonging to the same hierarchical k^{th} level. Hence, we are seeking a Nash Equilibrium solution in each level, starting by the highest level and working through till the lowest.

At each layer (k) of the game, we find that:

- By its definition, the game strategy space $\mathcal{C}$ is a convex, closed and bounded set.
- Furthermore, the cost function of each connection i $J_i^k(\mathbf{C}^k)$ is, whenever finite, continuous in c_i^l and convex in c_i^l for every fixed value of C^l [45].

Therefore, based on Theorem 2.2, the game possesses at least one Nash Equilibrium point [7].

It is worthy to note that in the above formulation, a player can be also thought of as a user (a major client, or a service provider, renting a portion of the network bandwidth). Once the NEP solution is reached, the user's equilibrium strategy c_i can be shared by various types of calls belonging to different traffic classes (e.g., voice, video, etc...). The partitioning of c_i among the different traffic classes is done by computing the *Contract Region* (Equation 2.9).

Chapter 4

System Implementation

The foundation of the present system being laid down in Chapter 3. from the entire perspective of the hierarchical network management approach to the problem formulation details of the VP partitioning and routing in the individual subnetworks. one can now proceed to discuss the practical issues involved with the actual implementation of the current system.

First. Section 4.1 begins by giving a detailed outline for the implementation steps of the hierarchical VP partitioning approach described in the previous chapter. In Section 4.2. an overview of the data structures used in building the network and traffic components is presented. Furthemore. an explaination of the interface between the current program and the optimization problem solver is provided. Section 4.3 outlines the formal description of the core routine explained in Section 4.2. Finally, some of the variation that are applied to the present model in order to test the performance under different working conditions are explained in Section 4.4.

4.1 Implementation of Hierarchical VP Allocation

In order to carry out the bandwidth allocation for Virtual Paths along with the desired analysis of some performance measures, we needed first to generate random topologies of large networks. The average number of nodes we are looking at is 250 nodes among which more than 70% are *external nodes*, that is, traffic-generating nodes, versus less than 30% are *internal nodes* or switches. The backbone network would contain, on the average, 50 links connecting the switches on top of which the whole network would contain 500 links carrying traffic to and from the external nodes. The following sections outline the steps of topology and traffic generation.

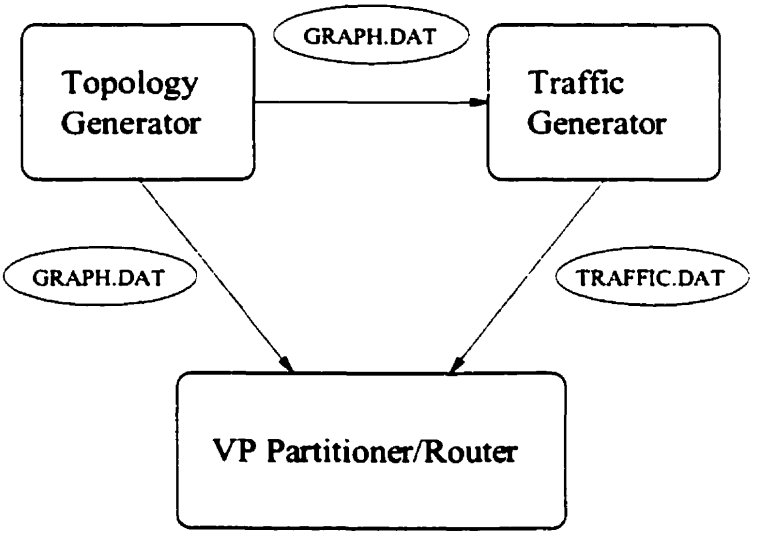


Figure 4.1: Relational Outline of the Simulator Modules

Figure 4.1 gives an illustrative overview of the main modules constituting the work at hand. The input to the *Random Topology Generator* is a set of parameters stated in Section 4.1.1. The output is a list of nodes and edges clustered in

hypernodes and hyperedges as described in Section 4.1.2.

The output file, *"GRAPH.DAT"* is fed into the *Traffic Generator* module whose task is to randomly generate a set of end-to-end connection demands with a random bandwidth request size. The parameters involved and the generation steps are given in Section 4.1.3.

The resulting file of the set of connections *"TRAFFIC.DAT"* together with the file *"GRAPH.DAT"* become the input to the core module, the *VP Partitioner/Router*, which solves the VP partitioning problem in the manner explained in Section 3.2. The resulting output is the set of connections with their routed demands, that is their VP allocation paths. Finally, the analysis of some performance measures is carried-out as will be shown in Chapter 5.

4.1.1 Physical Construction

The topology generator module comprises two main components: *(i)* the physical network construction, and *(ii)* the clustering resulting in the conceptual hierarchical structure of the global network. Sections 4.1.1 and 4.1.2 explain the two above components, respectively.

In the sequel, we outline the steps that were designed to randomly generate a large network topology. The set of constants and input parameters used by the module are specified below.

Constants:

- GRID: size of the two-dimensional square plane on which the physical network nodes are located.
- DELTA: minimum distance between any pair of nodes.

Input parameters:

- Initial number of switches $N_{s_original}$.
- Ratio of number of external nodes to number of switches R.
- Upper and lower capacity range values for backbone links.
- Upper and lower capacity range values for links connecting external nodes to switches.
- Proportion of extra links to be added N_{extra}.

The topology generation steps are as follows:

1. *Generate Initial Layout.* For the initial number of switches $N_{s_original}$, assign random coordinates in a two-dimensional plane, of GRID×GRID in size. A switch should have at least DELTA units of distance apart from all other switches.

2. *Cluster Switches.* Clustering of the initial set of switches to form the set of layer (2) nodes as described in Section 4.1.2.

3. *Create External Nodes.* For each hypernode in layer (2), generate N_e external nodes: where $N_e = R \times N_s$ and N_s is the number of switches in that hypernode. If N_s originally equals zero, generate 2 switch nodes and N_e becomes $N_e = 2R$. This is a design decision of insisting that at least two switches must exist in each hypernode so that transient traffic may have more than a single entrance/exit point.

4. *Connect Nodes.* First, the set of switches are connected by a spanning tree to ensure the existence of at least one path between any pair of switches [9]. This is performed instead of randomly connecting the switches then testing for their connectivity, i.e., that they form a connected graph. Second, the external nodes within each hypernode are randomly connected to the switches inside that hypernode. The last step is to ensure that no external nodes are left-out disconnected from the set of switches.

5. *Cluster Edges.* Within each hypernode, cluster edges as described in Section 4.1.2.

6. *Increase Connectivity.* Finally, add some random extra edges inside each hypernode to increase connectivity by a ratio of N_{extra} to the total number of nodes in the hypernode under consideration.

7. Repeat Clustering into higher levels in the same manner after incrementing the layer index by one.

4.1.2 Clustering and Hyper-Network Construction

Before describing the clustering technique implemented in the current system, the reader should note that, in the present work, there was no effort designated to achieve the best physical network partitioning that would yield the most favourable outcome in terms of network throughput. On the contrary, the clustering can be considered to be a given input to the current system. However, a simple clustering method was adopted here for the purpose of integrating a more comprehensive system. The clustering component can be substituted by a more involved clustering technique as an extension to this work. For instance, sophisticated circuit clustering

techniques can be found in [4] and [5] that can be suitably adapted to account for the communication network characteristics.

Depending on the scope of study, clusters can be regarded as large corporate networks or metropolitan networks. This in turn, will identify the nature of the upper clusters and so on. The number of layers is a design parameter. Three-layer networks are used in the present work to demonstrate the performance of various techniques.

The clustering procedure used in this work is simply based on the physical vicinity of nodes and edges. In essence, nodes and edges that are located within a given grid square are clustered into a hypernode in the following hierarchical layer. Whereas, edges connecting nodes pertaining to different parent nodes, are grouped in the following iteration of clustering. Graphical examples for the hyper-network layout are provided in Chapter 5.

For a given hypernode defined by a set of coordinates boundaries, clustering progresses in the following steps:

- *Cluster Nodes.* Group all nodes within the hypernode domain. The set of nodes have the same parent node and they form the underlying network of the hypernode.
- *Cluster Edges.* All edges that have both ends incident on nodes belonging to the same parent are added to the parent hypernode list of edges. Otherwise, their parent node will be determined by subsequent clustering operations.
- *Encode Topology.* All border nodes, defined by being connected to nodes not belonging to the same hypernode, are enumerated. A logical link connecting each pair of border nodes is constructed according to the approach described in Section 3.2.2.

4.1.3 Random Traffic Generation

The process of traffic generation is simple in its structure and has the following input parameters:

- Percentage of total fan of external nodes to be used in traffic generation.
- Percentage of number of connections to be generated in each layer.

The random traffic generation proceeds as follows:

- For each external node, calculate the total input/output fan, expressed by the sum of capacities of the links incident on that particular external node. The total amount of traffic generated to and from that node should not exceed a preset percentage of the total fan of the node. That percentage is a tunable input parameter. The default is 70%. This value is used for the experiments described in Chpater 5.
- For each hypernode, compile the set of external nodes. Keeping track of the actual bandwidth requests emanating from and ending at each external node, randomly select a pair of external nodes whose fan limits are not exceeded and assign a connection request of a random bandwidth size that does not exceed the minimum fan limit of the two nodes under consideration.
- The number of requests to be generated in each level is also determined by another input parameter. Typical values of these parameters are displayed in Chapter 5.
- As a final check, verify that all demands with the same source and destination nodes are grouped into a single demand with the collective bandwidth requests of the initial set of demands.

4.2 Data Exchange

The current work has been developed using the C++ programming language. A brief description of the data structures used is presented in Section 4.2.1 whereas the details of these data structures can be found in Appendix A. Section 4.2.2 deals with the interfacing details with the optimization solver Tomlab v.3.0 [27]. Tomlab is a collection of optimization routines that run in a Matlab environment [1].

4.2.1 Data Structures

The data structures used are implemented as classes with the object-oriented capabilities provided by the C++ programming language.

The telecommunication network is represented using three classes: *(i)* the "*Node*" class, *(ii)* the "*Edge*" class and *(iii)* the "*Graph*" class. Furthermore, a connection request is described by the "*Connection*" class. The source code (.h file) of these classes is given in Appendix A. A summary of their principal elements follows.

The Node Class

The *Node* class represents a simple node or a graph that is a hypernode in the network. For a simple node, the list of *children* is empty, whereas for the highest hypernode in the hierarchy, the *parent* node is null.

- Node attributes that are general to all nodes whether they are simple nodes or hypernodes:
  - Node identification number (a unique identifier).

[1] Matlab version 5.3 or higher.

- Node hierarchical level.
- Two-dimensional coordinates of the node.
- Border flag indicates if the node is a border node.
- Node type: internal (switch), external or hypernode.
- Node capacity: maximum flow that can pass through the node.
- Node status: current flow passing through the node.
- Parent: parent node id, if any.
- List of adjacent edges.

• Graph or hypernodes extra attributes:

- List of children nodes.
- List of children edges.
- List of children border nodes.
- List of their interconnecting logical links.

The Edge Class

As is the case with the *Node* class, the *Edge* class can represent a simple edge or a hyperedge.

• Edge general attributes:

- Edge identification number (a unique identifier).
- Edge hierarchical level.
- Source node from which the edge is forging out.

  - Destination node onto which the edge is terminating.
  - Edge type: physical, logical or hyperlink.
  - Edge capacity: maximum flow that the underlying physical link(s) can carry.
  - Edge status: current flow passing through the edge.
  - Parent: parent node id, if any.

- HyperEdge attributes:
  - List of children edges.

The Connection Class

Each connection request has the following attributes:

- Connection identification number (a unique identifier).
- Original Source: node requesting connection (level (1) node).
- Original Destination: destination node (level (1) node).
- Connection level.
- Source: current node acting as an intermediate source, either a hypernode or a level (1) switch.
- Destination: current node acting as an intermediate destination, either a hypernode or a level (1) switch.
- Amount of bandwidth requested.

- Connection average arrival rate. Connections arrival rate form a Poisson process with independent identically distributed exponential holding times with an average equal to a unit of time as is explained in Section 2.1.
- Average rate at which penalty is calculated or revenue is generated, depending on the mathematical model used.
- Quality of Service: an upper bound on the blocking probability of the connection.
- Parent: id of the parent connection that spawned the current connection. The value of this field is null for an original connection.
- Children: list of generated connections to be routed at subsequent levels of the optimization process.
- Routes: set of paths on which this connection is routed.

4.2.2 Interface with the Optimizer

For a given network configuration with a given set of demands, the optimization process in which demands are routed in their resulting VPs progresses in the manner depicted in Figure 4.2. Each loop consists of three main procedures:

1. *Prepare Traffic Data.* In this phase, the set of all traffic requests, the original ones along with the newly generated ones from previous iterations, are scanned and only the requests that fall within the current scope of the optimization are selected. The criteria by which a demand is selected is simply stated as follows: if both the source and destination nodes of the demand under consideration have the same (grand)parent node, and that parent node is

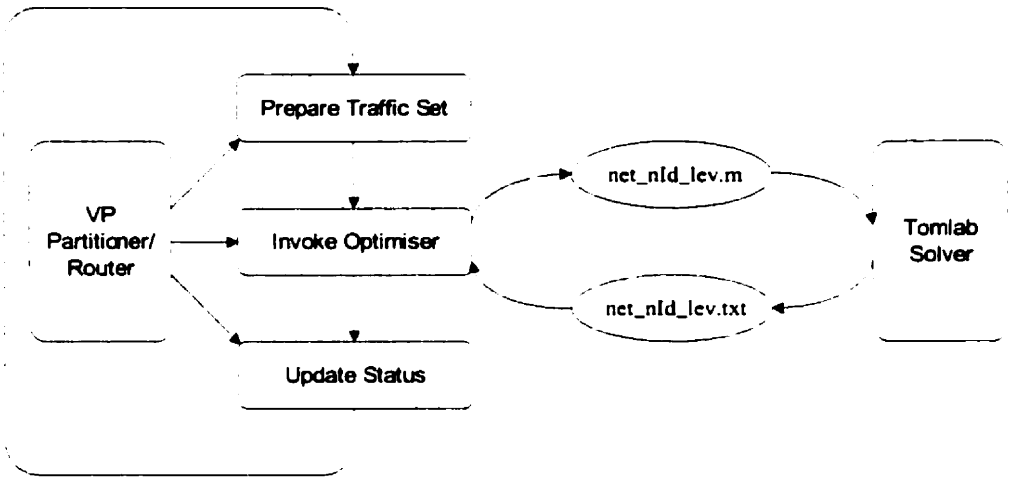


Figure 4.2: Operation of the VP partitioning and routing processor.

the hypernode for which the optimization problem is to be solved at this stage, then this demand is included in the list of current demands to be routed.

For instance, suppose node 115 is a hypernode in the network's second level of the hierarchical structure. Suppose also that the previous optimization iteration of its parent node, e.g. node 119, resulted in the generation of n demands, m of which are inherited by node 115 since their corresponding source and destination nodes have node 115 as their parent node. Therefore, all demands satisfying this criteria, including the m new demands are selected in the set of traffic demands that is to be routed by the next optimization iteration solving the VP partitioning and routing problem for node 115.

2. *Invoke Optimizer.* Based on the mathematical model described in Section 3.2.3, generate the ".m" file to be supplied to the Tomlab solver routine, *glcSolve* described in Section 4.4.1. Each *m_file* is named according to the following convention: *net_NodeId_IterationLevel.m*, where *NodeId* is the Node identifi-

cation number and *IterationLevel* is the current level at which the optimizer is running. For example, if the hypernode under consideration is node 115 and the set of traffic to be routed is the set of level (1) connections, then the input file name becomes *net_115_0.m*[2]. A sample *m_file* can be found in Appendix B.

Once the Tomlab routine has completed its run, it generates an output file with the same naming convention: *net_NodeId_IterationLevel.txt*. So for the same example mentioned above, a text file called *net_115_0.txt* is generated. That file contains the paths on which each connection is routed along with the amount of bandwidth allocated to that path.

3. *Update Status*. The third and final stage in each iteration is responsible for updating both the network and the traffic status. This is the most challenging procedure to develop due to the fact that it has to be both: *(i)* as generic as possible to deal with any number of hierarchical layers, *(ii)* as well as being able to get recursively deeper into each hypernode details to explore the possibilities of the underlying physical paths for a given connection. A formal description of this core function is given in Section 4.3. It progresses in the main following steps:

   - Update status of each node and each link to reflect the new amount of flow passing through it. This is done in a straightforward manner by adding the resulting amount of bandwidth for a given connection c_i to the current status of all the physical links forming the path on which it is routed. Consequently, the status of the hyperedges and hypernodes are updated.

[2]In the actual implementation, the levels are indexed beginning with zero.

- For each connection, for each route specified by hyperedges, translate the routing details by determining which children edges are used with how much bandwidth.

  For instance, suppose that hyperedge E_{12} has 3 children (hyper)edges E_1, E_2 and E_3, and the amount of bandwidth to be routed through E_{12} is equal to 5 units of bandwidth. Furthermore, suppose that each (hyper)edge of the 3 children has a total available capacity that is greater than 5. According to the current implementation, a *fairness* in distribution is adopted in which each edge will get 33% portion of the 5 units, that is 2, 2 and 1 unit for the 3 edges respectively, thus following a uniform distribution.

  Another alternative is to distribute the bandwidth in a *greedy* manner, that is to assign the 5 units of bandwidth to the child edge having the minimum available capacity that can accommodate it. However, we chose the first alternative believing that, with a fair distribution, the network will tend to have less saturated or congested links. The second alternative constitutes a possible extension to the work presented here.

- For a route consisting of hyperedges, for any intermediate hypernode, recursively pass-down a connection request with the proper bandwidth required through the relevant source-destination pair. By keeping track of the underlying border nodes, specify the set of input nodes and output nodes depending on the traffic flow direction. Specify also the set of common nodes, that is, the intersection of both the input and output nodes sets. Figure 4.3 illustrates one such possible scenario:

  - For the hypernode shown in Figure 4.3, there are 3 border nodes.
  - According to the illustrated traffic pattern, the border nodes are cat-

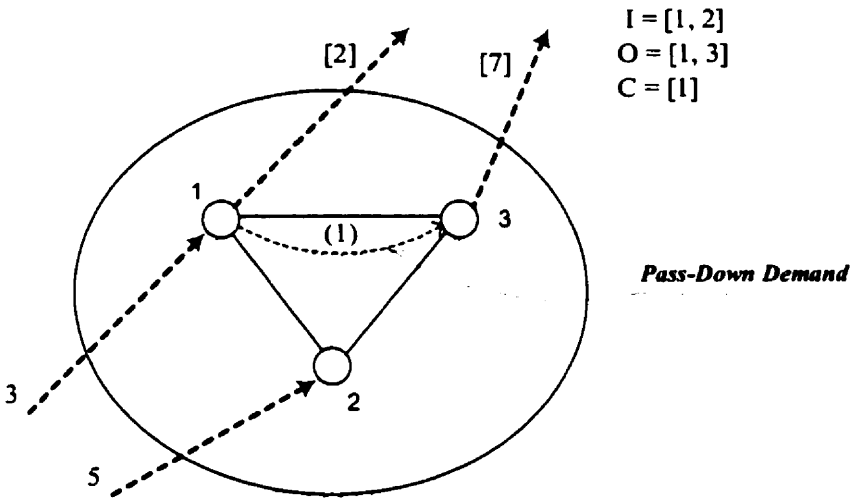


Figure 4.3: Traffic flow in an intermediate node.

egorized into 3 sets: I. O. and C being Input. Output and Common set. respectively.

- For each node in the Common set. do the following: *(i)* calculate the total fan-in (3 for node 1) and total available fan-out ([2]) in this case. *(ii)* if (fan-in $\leq$ fan-out) then pass all fan-in through common node 1; otherwise. issue a pass-down request of routing the (fan-in - fan-out) units of bandwidth from the common node 1 to the output node 3.

The previous example gives a flavour of the global operation of the "*Intermediate-Node*" algorithm outlined formally in the following section.

4.3 Processing of Routed Connections

In this section, a pseudo-code like description of the "*Update-Status*" explained in the previous section is presented. As a reminder of the set-up in which this function operates, the reader should observe that this function processes the output results resulting from the optimization solver (Figure 4.2).

Update-Status

For each connection i in solved set of connections do:

For each route $r \in R$ do: % R: set of routes of i

For each link $l = 1 to L$ do: % L: set of links for route r

if(l is first)

then

previous-link = NULL

else

previous-link = link($l - 1$)

EndIf

if(l is last)

then

next-link = NULL

else

next-link = link($l + 1$)

EndIf

% set the proper node between previous-link and next-link

set current node = $n_{current}$

set pass-down-list = {}

Fan-Node($n_{current}$, previous-link, next-link, Pass-Down-List)

```
            Add pass-down-list to current set of unresolved demands
        End % For l
    End % For each r
End % For each connection
```

Fan-Node($n_{current}$, previous-link, next-link, Pass-Down-List)

```
if( previous-link == NULL )
    Partition bandwidth demand among children of next-link % and grandchildren
    Define s as the Original Source for the current connection
    s = Parent(s) until level(s) = level(n_current) - 1
    For each child-edge e do:
        If( bw(e) > 0 AND (S(e) != s AND T(e) != s) AND
            ( Parent(s) == Parent(S(e)) OR Parent(s) == Parent(T(e)) ) )
        then
            if( Parent(s) == Parent(S(e)) )
            then
                d = S(e)
            else
                d = T(e)
            EndIf
            Add connection (s. d) with bandwidth bw(e) to Pass-Down-List
        EndIf
    End % For each child-edge
EndIf
```

```
if( next-link == NULL )
    Define d as the Original Destination for the current connection
    d = Parent(d) until level(d) = level(n_current) - 1
    For each child-edge e do:
        If( bw(e) > 0 AND (S(e) != d AND T(e) != d) AND
            ( Parent(d) == Parent(S(e)) OR Parent(d) == Parent(T(e)) ) )
        then
            if( Parent(d) == Parent(S(e)) )
            then
                s = S(e)
            else
                s = T(e)
            EndIf
            Add connection (s, d) with bandwidth bw(e) to Pass-Down-List
        EndIf
    End % For each child-edge
EndIf
if( previous-link != NULL AND next-link != NULL )
then
    Intermediate-Node(n_current, previous-link, next-link, Pass-Down-List)
EndIf
```

Intermediate-Node($n_{current}$, previous-link, next-link, Pass-Down-List)

Compose the three sets I, O and C

```
For each node n in C do:
    if( level(n) > 1 ) % n is also a hypernode
    then
        Intermediate-Node(n, previous-link, next-link, Pass-Down-List)
    else
        Calculate total fan-in to n as the sum of flow entering n
        Define next-children to be the set of children edges of next-link
        that have n as one of their ends
        Calculate total fan-out as the sum of available capacity
        of all edges in next-children
        if( fan-in <= fan-out )
        then
            Partition fan-in on the corresponding set of
            edges in next-children
        else
            Partition amount = fan-out on the corresponding set of
            edges in next-children
            Select a random output-node o_n from the set O that
            is not equal to n and route the amount = (fan-in - fan-out)
            through o_n
            Add connection (n, o_n) with bandwidth = (fan-in - fan-out)
            to Pass-Down-List
        EndIf
    EndIf
End % For each node
```

```
For each node n in I and not in C do:
    Calculate total fan-in to n as the sum of flow entering n
    repeat = True
    while( repeat )
        Select a random node o_n from the set O
        Define next-children to be the set of children edges of next-link
        that have n as one of their ends
        Calculate total fan-out as the sum of available capacity
        of all edges in next-children
        if( fan-in <= fan-out )
        then
            repeat = False
            Partition fan-in on the corresponding set of
            edges in next-children
            Add connection (n. o_n) with bandwidth = (fan-in)
            to Pass-Down-List
        else
            fan-in = fan-in - fan-out
            repeat = True
        EndIf
    EndWhile
End % For
```

In the previous functions, the *Partition* statement is an implementation of a function that partitions a given amount uniformely into a predefined number of elements. It takes into consideration whether the portion of each element is an

integer or a real positive number depending on the mathematical model that is being used.

4.4 Implementation Variants

Other than the two alternatives mentioned in the previous section related to the bandwidth distribution among children edges, there are other variants to the current implementation that are described below.

1. *Mathematical model variation.* Based on the problem formulation given in Section 3.2.3, for the same set of parameters and variables, an implementation of a classical VP partitioning problem model can be stated as follows:

   From the subnetwork point of view, the local network manager tries to maximize the total revenue. Assuming that for each demand, there is an associated revenue α_i\$/unit-time. The total revenue generated by all connections passing through the subnetwork will be:

   $$\max Rev = \sum_{k=1}^{K} \sum_{i \in \mathcal{I}_k} \lambda_i \alpha_i [1 - L_i] \tag{4.1}$$

   where L_i is given by Equation (2.7):

   $$L_i = 1 - \prod_{l \in r_i} (1 - \mathcal{E}_l)$$

   subject to

   $$C^l \leq B^l \qquad \forall l \in L,$$

   $$\mathcal{E}(\rho_i, c_i) \leq \kappa_i \qquad \forall i \in \mathcal{I}$$

   Note that the revenue does not depend on a specific route just on the source-destination node pair of the connection. In general, connections in higher

hierarchical levels generate more income than those in lower levels. The reason for this statement is that on the average, the higher the connection level is, the greater is the number of links used to route its demand.

2. *Order of optimization.* Instead of following the Stackelberg game model described in the previous chapter, a Nash (fair) model can be applied and as well as a "reverse" Stackelberg model. Hence, instead of a multi-level optimization in which one solves the routing problem for the connections in the highest level, the leaders, followed by the next level and so on, all connections are treated equally regardless of what level they belong too as in a Nash game scenario. Another alternative is also tested in which the original order is reversed and the optimization starts with the level (1) connections, followed by level (2) and so on up to the highest level. That is, instead of just a top-down approach, the equal and bottom-up orders of optimization are tested as well.

3. *Domain variation.* As a final variant, most of the problems were solved not just in the discrete domain, but also in the continuous one. In the first case, the amount of bandwidth allotted to each VP had to be an integer value, while in the second case, this condition is relaxed. For the discrete domain, the *glcSolve* routine (Section 4.4.1) in the Tomlab library is used. The *glcSolve* routine is based on the DIRECT method [30] described in Section 4.4.2.

   It is worthy to note that, for the same problem, the number of variables in each optimization level is increased in the continuous case versus the discrete one. This follows naturally from the fact that each connection tends to use more routes with smaller bandwidth in the continuous case than in the discrete case where the bandwidth of each route is quantized on the set of positive integers.

Due to the fact that the performance of the DIRECT method worsens as the number of variables is increased, as in the continuous case, another solver was used in the continuous domain experiments, namely, the *"conSolve"* solver which is based on quadratic curve fitting, or quadratic programming [26, 49] to reach the global optimum. A brief introduction to quadratic programming is given in Section 4.4.3.

4.4.1 The Discrete Optimization Solver: *"glcSolve"*

The routine *glcSolve* implements an extended version of the DIRECT method presented in [30] that handles problems with both nonlinear and integer constraints [27].

DIRECT is a modification of the standard Lipschitzian approach that eliminates the need to specify a Lipschitz constant. Since no such constant is used, there is no natural way of defining convergence, except when the optimal function value is known. Therefore, *glcSolve* is run for a predefined number of function evaluations and considers the best function value found as the optimal one. There is also the option to restart *glcSolve* with the final status of all parameters from the previous run [27]. In general, *glcSolve* solves global mixed-integer nonlinear programming problems of the form:

$$\min_x f(x)$$

subject to:

$$\begin{array}{ccccc} x_L & \leq & x & \leq & x_U \\ b_L & \leq & Ax & \leq & b_U \\ c_L & \leq & c(x) & \leq & c_U \\ x_i & & \text{integer} & & i \in \text{Integers} \end{array}$$

where $x, x_L, x_U \in \mathcal{R}^n$, $c(x), c_L, c_U \in \mathcal{R}^{m_1}$, $A \in \mathcal{R}^{m_2 \times n}$ and $b_L, b_U \in \mathcal{R}^{m_2}$.

Input Description

x_L, x_U	Lower and upper bounds for x, respectively. Both must be given to restrict the search space.
A	Constraint matrix for linear constraints.
b_L, b_U	Lower and upper bounds on the linear constraints, respectively.
c_L, c_U	Lower and upper bounds on the general constraints, respectively.
Integers	Set of integer variables.

Other inputs include the file name for computing the objective function $f(x)$ and the filename for computing the vector of constraint functions $c(x)$. The number of maximum function evaluations (default: 200), the weight parameter ϵ (default: 10^{-4}) are also given as input parameters to the routine *glcSolve*. The weight parameter ϵ defines the optimality percentage, i.e. the final solution is $100\epsilon\%$ away from the optimal solution. For $\epsilon = 10^{-4}$, the best solution is 0.01% from the optimal solution.

Output Description

feasible	Flag indicating if a feasible point has been found.
Iter	Number of iterations.
FuncEv	Number of function evaluations.
f_k	Best function value found.
x_k	Matrix with all points giving the function value f_k.
c_k	Nonlinear constraints values at x_k.

In the case of non-discrete domain solutions, the set of *Integers* in the *glcSolve* is an empty set.

4.4.2 The DIRECT Global Optimization Algorithm

The DIRECT method [30] is based on the DIvide RECTangles approach that is an extension and modification of the original work of Shubert's global optimization algorithm [65].

Consider the problem of finding the global minimum of a function $f(x)$ defined on the closed interval $[l, u]$. Standard Lipschitzian algorithms assume that there exists a finite bound on the rate of change of the function: that is, they assume that there exists a positive K, the Lipschitz constant, such that

$$|f(x) - f(x')| \leq K|x - x'|. \qquad \forall x, x' \in [l, u] \tag{4.2}$$

Based on this assumption, a lower bound can be placed on the function in any closed interval whose endpoints have been evaluated. Figure 4.4 illustrates one such function and the steps followed by Shubert's algorithm to search for the global minimum or minima of the function $f(x)$. In the process of evaluating the endpoints of the interval $[l, u]$, the following two inequalities must be satisfied:

$$f(x) \geq f(l) - K(x - l) \tag{4.3}$$

$$f(x) \geq f(u) + K(x - u) \tag{4.4}$$

These inequalities correspond to the two lines with slopes $-K$ and $+K$ in Figure 4.4 (a). The lowest value that $f(x)$ can attain occurs at the intersection of these two lines, that is at the bottom of the V that forms the lower bound of the

function $f(x)$ in the interval $[l, u]$. Let us denote this point by $X(l, u, f, K)$ and the corresponding lower bound of f by $B(l, u, f, K)$:

$$X(l, u, f, K) = \frac{l+u}{2} + \frac{[f(l) - f(u)]}{2K} \tag{4.5}$$

$$B(l, u, f, K) = \frac{[f(l) + f(u)]}{2} - \frac{K(u-l)}{2} \tag{4.6}$$

These two equations form the core of Shubert's algorithm that can be summarized in the following steps:

1. Initialize the search by evaluating the endpoints l and u as shown in Figure 4.4 (a).

2. Evaluate the function at $x_1 = X(l, u, f, K)$. This divides the search space into two intervals, $[l, x_1]$ and $[x_1, u]$ (part (b)).

3. Determine which interval to explore next by choosing the interval with the lowest B-value. In case of a tie (as is this case), choose arbitrarily any of the equivalent intervals.

4. If the minimum found is within some prespecified tolerance of the current best solution, then stop. Otherwise go to step (2).

Figure 4.4 (c) demonstrates one extra step after which the interval $[x_1, u]$ is the next search space to be explored since it has the lowest lower bound of all three intervals. At any point in Shubert's algorithm, the V's for all the intervals form a piecewise linear function that approximates $f(x)$ from below. In Equation 4.6, the first term is lower (and therefore better when we minimize) when the function values at the endpoints are low. Therefore, this term leads to the selection of intervals where previous function evaluations have been good, that is, it leads to do local search.

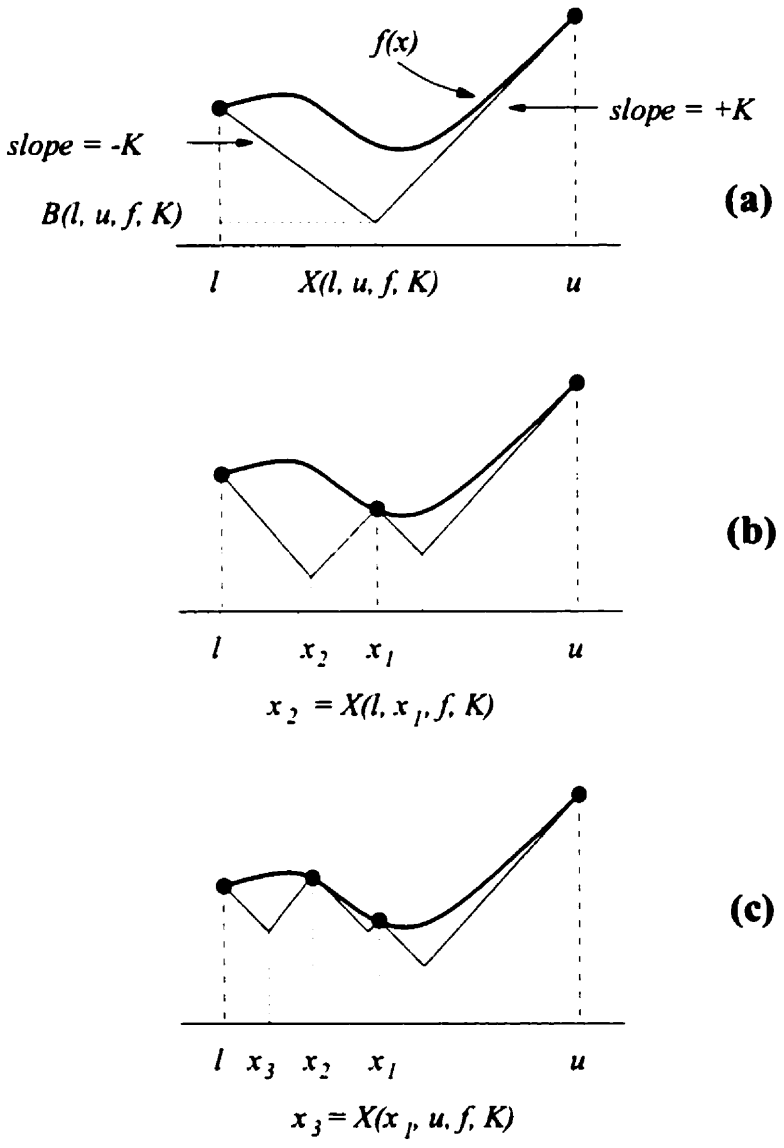


Figure 4.4: Shubert's algorithm

The second term, on the other hand, is lower algebraically as the interval gets bigger, therefore, this term leads to the selection of intervals with large amount of unexplored territory, that is, it leads to do a global search. The Lipschitz constant K serves as a relative weight on global versus local search.

The Shubert's algorithm described above suffers from two main problems:

1. Slow convergence. Since the Lipschitz constant must be an upper bound on the rate of change of the function, it will generally be quite high. Shubert's algorithm places a high weight on global search, consequently, it exhibits slow convergence. Once the basin of convergence of the optimum is found, the search would proceed more quickly if K could be reduced. In practice, however, it is difficult to know when and how to reduce K. Thus, K is left unchanged and slow convergence is inevitable.

2. The number of function evaluations during the initialization phase. Though two functions evaluations at the endpoints of a unidimensional problem form no real inconvenience, the 2^n vertices of multi-dimensional problems has a high computational complexity.

The DIRECT algorithm was conceived to overcome these shortcomings in the Shubert's algorithm. Other algorithms were also implemented with the same goal such as [20, 56, 61].

4.4.2.1 DIRECT Algorithm in One Dimension

In order to make Shubert's algorithm practical in higher dimensions, the DIRECT method partitions the search space in a different way. Instead of evaluating the function at the endpoints of an interval, it is only evaluated at the center point of

the interval in question. In n dimensions, this translates to initializing the algorithm by sampling just one point (the center of the search space) as opposed to all 2^n vertices of the space.

This modification is accompanied by some other changes. Equations 4.3 and 4.4 for the continuous interval $[l, u]$ over the function $f(x)$ become:

$$f(x) \geq f(c) + K(x - c) \qquad x \leq c \tag{4.7}$$

$$f(x) \geq f(c) - K(x - c) \qquad x \geq c \tag{4.8}$$

where c is the centerpoint of the interval, that is, $c = (l + u)/2$. In Figure 4.5, these inequalities correspond to the lines with slopes $+K$ and $-K$ and the function must lie above the inverted V formed by their intersection. The lowest value the function can attain occurs at the endpoints l and u, and has the value

$$B = f(c) - K(u - l)/2 \tag{4.9}$$

The interval is then divided into thirds and the function is evaluated at the center

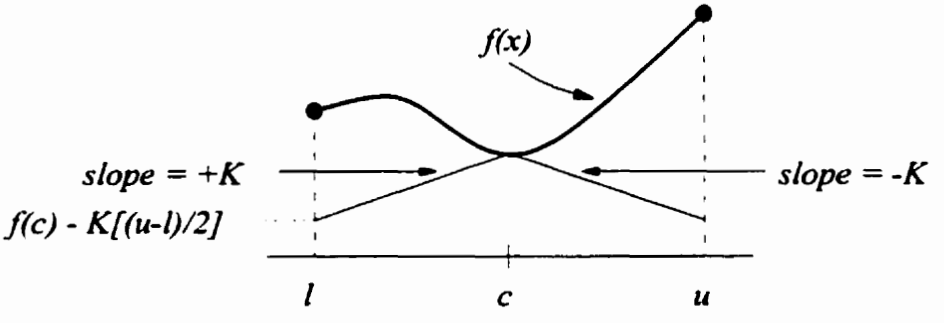


Figure 4.5: Computing a lower-bound with centerpoint sampling.

points of the left and right thirds. The original center point simply becomes the center of a smaller interval. Suppose now that at some iteration point, there exists m intervals $[a_i, b_i]$, $i = 1, \ldots, m$ with corresponding center points $c_i = (b_i - a_i)/2$. In

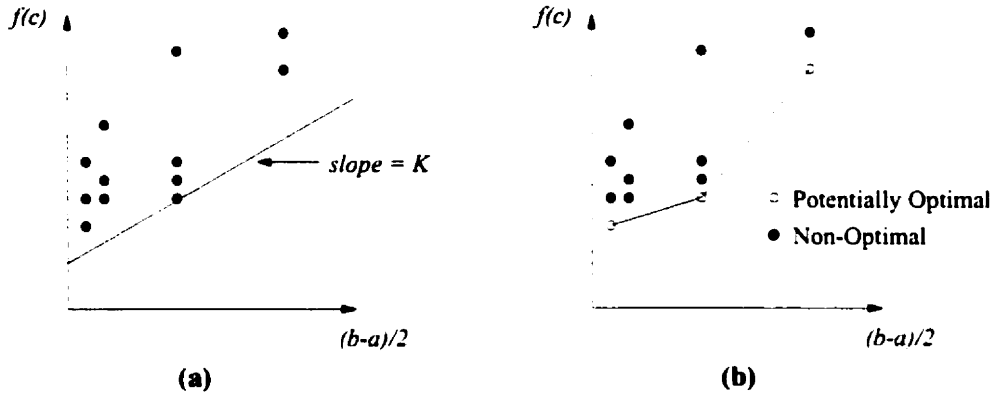


Figure 4.6: Set of potentially optimal intervals.

Figure 4.6, each interval is represented by a single dot with horizontal coordinate c_i and vertical coordinate $f(c_i)$. This representation possesses the following features:

- The horizontal coordinate is the distance from the interval's center to its vertices. It captures the goodness of the interval with respect to global search, that is, the goodness based on the amount of unexplored territory in the interval.

- The vertical coordinate is the value of the function at the interval's center. It captures the goodness of the interval with respect to local search, that is, goodness based on known function values.

- If one passes a line with slope K through any in the diagram illustrated in Figure 4.6 (a), the vertical intercept will be the lower bound for the corresponding interval.

The Lipschitz constant, reflected in the slope of the line, determines the relative weighting of global versus local search, and is generally overemphasizing the global search. In the DIRECT method, instead of assigning a single weight K for the global

search, all possible relative weights are considered as is depicted in Figure 4.6 (b). This corresponds to identifying the set of "potentially optimal" intervals according to the following definition:

Definition 4.1 *Suppose that the interval* $[l, u]$ *is partitioned into intervals* $[a_i, b_i]$ *with midpoints* c_i *for* $i = 1, \ldots, m$. *Let* $\epsilon > 0$ *be a positive constant, and the* f_{min} *be the current best function value. Interval* j *is said to be potentially optimal if there exists some rate-of-change constant* $\tilde{K} > 0$ *such that*

$$f(c_j) - \tilde{K}[(b_j - a_j)/2] \leq f(c_i) - \tilde{K}[(b_i - a_i)/2] \qquad \forall i = 1, \ldots, m.$$

$$f(c_j) - \tilde{K}[(b_j - a_j)/2] \leq f_{min} - \epsilon|f_{min}|$$

The first condition forces the interval to be on the lower right of the convex hull of the dots. The second condition insists that the lower bound for the interval, based on the constant $\tilde{K}$, exceed the current best solution by a nontrivial amount. This second condition is needed to prevent the algorithm from becoming too local in its orientation. It was shown that the DIRECT algorithm is fairly insensitive to the setting of ϵ [30]. A final note before proceeding to the multivariate algorithm is that the order of selecting the potentially optimal intervals is irrelevant as long as they are all selected.

4.4.2.2 DIRECT Algorithm in Multiple Dimensions

The main issue in extending DIRECT to several dimension concerns how to partition the search space of n variables. Without loss of generality, the lower and upper bounds on every variable can be normalized to zero and one, respectively, thus resulting in the search space represented by the n-dimensional unit hypercube.

As the algorithm proceeds, this hypercube is partitioned into hyperrectangles, each with a sampled point at its center. Before describing the DIRECT algorithm in its mulivariate format, let us extend Definition 4.1 for the single dimension case to the multiple variable case as follows:

Definition 4.2 *Suppose that a partition of the unit hypercube resulted into m hyperrectangles. Let $\mathbf{c}_i$ denote the center point of the i^{th} hyperrectangle, and let d_i denote the distance from the center point to the vertices. Let $\epsilon > 0$ be a positive constant. A hyperrectangle j is said to be potentially optimal if there exists some $\tilde{K} > 0$ such that*

$$f(\mathbf{c}_j) - \tilde{K}d_j \leq f(\mathbf{c}_i) - \tilde{K}d_i \qquad \forall i = 1, \ldots, m.$$

$$f(\mathbf{c}_j) - \tilde{K}d_j \leq f_{min} - \epsilon|f_{min}|$$

Multivariate DIRECT Algorithm

1. Normalize the search space to be the unit hypercube. Let $\mathbf{c}_1$ be the centerpoint of this hypercube and evaluate $f(\mathbf{c}_1)$. Set $f_{min} = f(\mathbf{c}_1), m = 1$ and $t = 0$ (iteration counter).

2. Identify the set S of potentially optimal rectangles.

3. Select any rectangle $j \in S$.

4. Using the procedure for dividing rectangles, determine where to sample within rectangle j and how to divide the rectangle into subrectangles. Update f_{min} and set $m = m + \Delta m$, where Δm is the number of new points sampled.
   Dividing Rectangle Procedure:

(a) Identify the set I of dimensions with the maximum side length. Let δ equal one-third of this maximum side length.

(b) Sample the function at the points $\mathbf{c} \pm \delta \mathbf{e}_i$ for all $i \in I$, where $\mathbf{c}$ is the center of the rectangle and $\mathbf{e}_i$ is the i^{th} unit vector.

(c) Divide the rectangle containing $\mathbf{c}$ into thirds along the dimensions in I, starting with the dimension with the lowest value of

$$w_i = \min\{f(\mathbf{c} + \delta \mathbf{e}_i), f(\mathbf{c} - \delta \mathbf{e}_i)\}$$

and continuing to the dimension with the highest w_i.

5. Set $S = S - \{j\}$. If $S \neq \{\}$ go to Step 3.

6. Set $t = t + 1$. If $t = T$, stop; the iteration limit has been reached. Otherwise, go to Step 2.

An illustrative example of how the rectangles are divided in the DIRECT algorithm is shown in Figure 4.7, in which solution (b) is favoured over solution (a) in the depicted two-dimensional situation. The function value at each point is recorded for the sake of illustration.

Comparison of the DIRECT algorithm to other global optimization techniques was carried out and it was shown in [30] that DIRECT outperformed its competitors in speed of convergence and the number of function evaluations.

4.4.3 The Continuous Optimization Solver: "*conSolve*"

$$\min_x f(x)$$

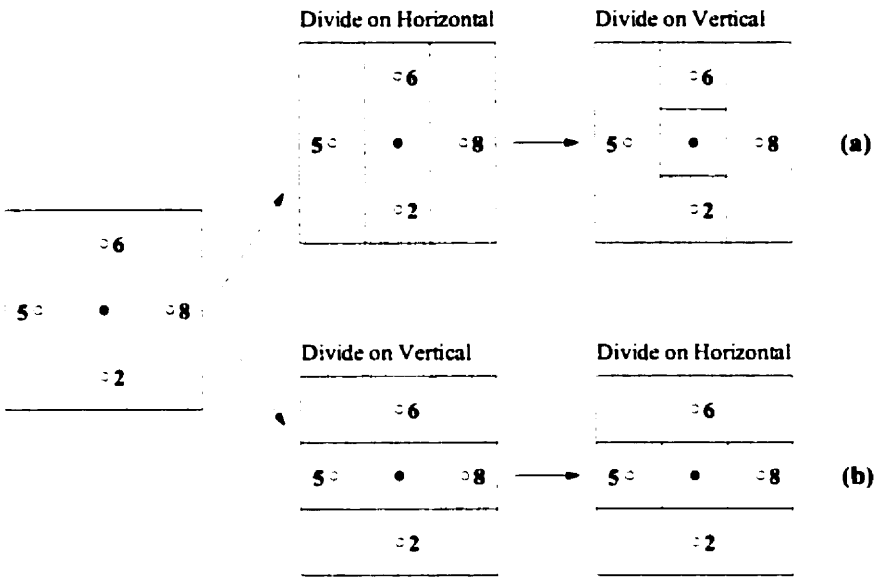


Figure 4.7: Dividing rectangles in the DIRECT algorithm.

subject to:

$$\begin{array}{ccccc} x_L & \leq & x & \leq & x_U \\ b_L & \leq & Ax & \leq & b_U \\ c_L & \leq & c(x) & \leq & c_U \\ x_i & & \text{integer} & & i \in \text{Integers} \end{array}$$

where $x, x_L, x_U \in \mathcal{R}^n$, $c(x), c_L, c_U \in \mathcal{R}^{m_1}$, $A \in \mathcal{R}^{m_2 \times n}$ and $b_L, b_U \in \mathcal{R}^{m_2}$. [27].

Input Description

x_L, x_U	Lower and upper bounds for x, respectively. Both must be given to restrict the search space.
A	Constraint matrix for linear constraints.
b_L, b_U	Lower and upper bounds on the linear constraints, respectively.
c_L, c_U	Lower and upper bounds on the general constraints, respectively.
x_0	Starting point.

Other inputs include the file name for computing the objective function $f(x)$ and the filename for computing the vector of constraint functions $c(x)$. The file names for the gradient vector $g(x)$ and the Hessian matrix $H(x)$ can also be supplied as input to the solver. The number of maximum function evaluations (default: 200) can be tuned by setting it to a new value and adding it to the input set. The detailed description of the *conSolve* routine can be found in [27].

Output Description

$Iter$	Number of iterations.
$FuncEv$	Number of function evaluations.
f_k	Best function value found.
x_k	Matrix with all points giving the function value f_k.
c_k	Nonlinear constraints values at x_k.
g_k	Gradient value at x_k.
H_k	Hessian value at x_k.
v_k	Lagrange multipliers at x_k.

The *conSolve* routine implements two sequential quadratic programming (SQP) algorithms for general constrained minimization problems. A brief introduction to quadratic programming is presented here.

Consider the nonlinear optimization problem described by:

$$\begin{array}{lll} \text{Minimize:} & f(\mathbf{x}) & \\ \text{Subject to:} & h_i(\mathbf{x}) = 0 \quad & i = 1, \ldots, m \\ & g_j(\mathbf{x}) \geq 0 \quad & j = 1, \ldots, p \end{array} \tag{4.10}$$

This problem can be modified by replacing the nonlinear functions of problem (4.10)

by their first-order Taylor series approximates at the point $\mathbf{x}_k$ to yield:

$$\begin{array}{lll} \text{Minimize:} & f(\mathbf{x}_k) + \nabla^T f(\mathbf{x}_k)(\mathbf{x} - \mathbf{x}_k) & \mathbf{x} \in E^n \\ \text{Subject to:} & h_i(\mathbf{x}_k) + \nabla^T h_i(\mathbf{x}_k)(\mathbf{x} - \mathbf{x}_k) = 0 & i = 1, \ldots, m \\ & g_j(\mathbf{x}_k) + \nabla^T g_j(\mathbf{x}_k)(\mathbf{x} - \mathbf{x}_k) \geq 0 & j = 1, \ldots, p \end{array} \tag{4.11}$$

where $\nabla f(\mathbf{x}_k)$ is the $(n \times 1)$ gradient *column* vector of first-order partial derivatives of f at $\mathbf{x}_k$, defined as:

$$\frac{\partial f}{\partial x_i} \qquad \mathbf{x} \in E^n, i : 1, \ldots, n.$$

The functions $f(\mathbf{x}_k), \nabla f(\mathbf{x}_k), h_i(\mathbf{x}_k), \nabla h_i(\mathbf{x}_k), g_j(\mathbf{x}_k)$ and $\nabla g_j(\mathbf{x}_k)$ are all constant vectors or scalars evaluated at $\mathbf{x}_k$. Therefore, problem (4.11) is a linearized version of the nonlinear programming problem (4.10) [26, 49].

Suppose now the Taylor series is expanded to include the second-order approximates. Let us start by defining:

$$q(\mathbf{x}) = f(\mathbf{x}_k) + \nabla^T f(\mathbf{x}_k)(\mathbf{x} - \mathbf{x}_k) + \frac{1}{2}(\mathbf{x} - \mathbf{x}_k)^T \mathbf{F}(\mathbf{x}_k)(\mathbf{x} - \mathbf{x}_k)$$

to be the quadratic function that agrees with, or approximates $f(\mathbf{x})$ at $\mathbf{x}_k$ up to second derivatives. From elementary calculus, the function $q(\mathbf{x})$ has a local extremum point (maximum or minimum) at a point $\mathbf{x}_o$ if:

$$\nabla q(\mathbf{x}_o) = 0$$

and $Q(\mathbf{x}_o)$ is positive definite (minimum) or negative definite (maximum) where Q is the $(n \times n)$ matrix formed by the second-order partial derivatives.

This results in:

$$\nabla q(\mathbf{x}) = \nabla f(\mathbf{x}_k) + \mathbf{F}(\mathbf{x}_k)(\mathbf{x} - \mathbf{x}_k) = 0$$

where $\mathbf{F}$ is the $(n \times n)$ Hessian matrix of second-order partial derivatives, whose $(i,j)^{th}$ element has the form:

$$\frac{\partial f}{\partial x_i \partial x_j} \qquad \mathbf{x} \in E^n, i,j : 1,\ldots,n.$$

We may then calculate an estimate $\mathbf{x}_{k+1}$ of the minimum point of f (since we are minimizing) by finding the point where the derivative of q vanishes, we obtain:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \mathbf{F}^{-1} \nabla f(\mathbf{x}_k) \tag{4.12}$$

The iterative equation, Equation (4.12), is the Newton method to minimize the function f. Basically, the Newton method fits a quadratic curve approximation of the function to be minimized, and finds the next point by setting it equal to the minimum of this fitted quadratic function.

Other forms of quadratic functions can also be used [26, 49]. A graphical illustration of approximating a nonlinear function by a quadratic one is given in Figure 4.8.

To account for the constraints, a *Langrangian* augmentation of the objective function is applied. That is, the new objective function becomes:

$$\text{Minimize:} \quad l(\mathbf{x}, \lambda, \mu) = f(\mathbf{x}) + \lambda \mathbf{h}(\mathbf{x}) + \mu \mathbf{g}(\mathbf{x}) \tag{4.13}$$

This transforms the original constrained nonlinear problem (4.10) into an unconstrained nonlinear version with $n + m + p$ variables. The study of the necessary and sufficient conditions that guarantee that the local minimum is reached at a certain point $\mathbf{x}^*$ is explained in [49].

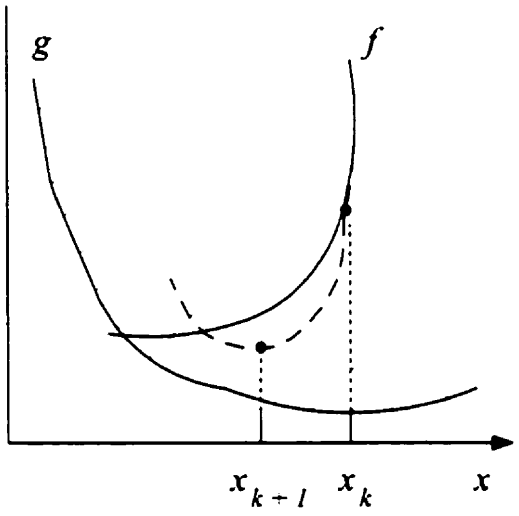


Figure 4.8: Newton's method for minimization.

Chapter 5

Analysis and Experimental Results

In this chapter, the experimental simulations that were conducted are explained and the performance measures resulting from these experiments are reported. Once the input parameters for the network and traffic generation phase were supplied, the resulting "*GRAPH.DAT*" and "*TRAFFIC.DAT*" were fed to the VP partitioning and routing processor. The resulting output is the set of routed connections found in "*Routed_TRAFFIC.DAT*". All experiments reported in this chapter were run on a Pentium-300 processor with 124 MB RAM.

5.1 Experimental Comparison

Three different networks were generated and used for testing the current system. Figure 5.1 gives a graphical illustration for Network (1). Network (2) and Network (3) are illustrated in Figure 5.2 and Figure 5.3, respectively. The data for the

three networks is summarized below:

Network Data	Net (1)	Net (2)	Net (3)
Number of Level (1) Nodes:	99	295	432
Number of HyperNodes (Figure 5.4):	21	21	21
Number of Switches:	33	59	72
Number of External Nodes:	66	236	360
Total Number of Nodes:	120	316	453
Number of Level (1) Edges:	183	547	783
Number of HyperEdges:	47	56	52
Total Number of Edges:	230	603	835

The clustering of these physical networks followed the algorithm described in Section 4.1.2. and was solely based on the geographical location of the nodes. Other clustering criteria could be used instead as was mentioned. but for the sake of simplicity and to focus on the main axiom of this research. the technique described in Section 4.1.2 was adopted. The resulting upper conceptual layers are depicted in Figure 5.4

During the traffic generation phase. the following parameters were used to generate three different sets of requests for any given network. In each set. the number of level (i) connections generated is proportional to the number of external nodes existing in the hypernode they are generated in. as was explained in Section 4.1.3. The ratio is given by the corresponding entry in the following table:

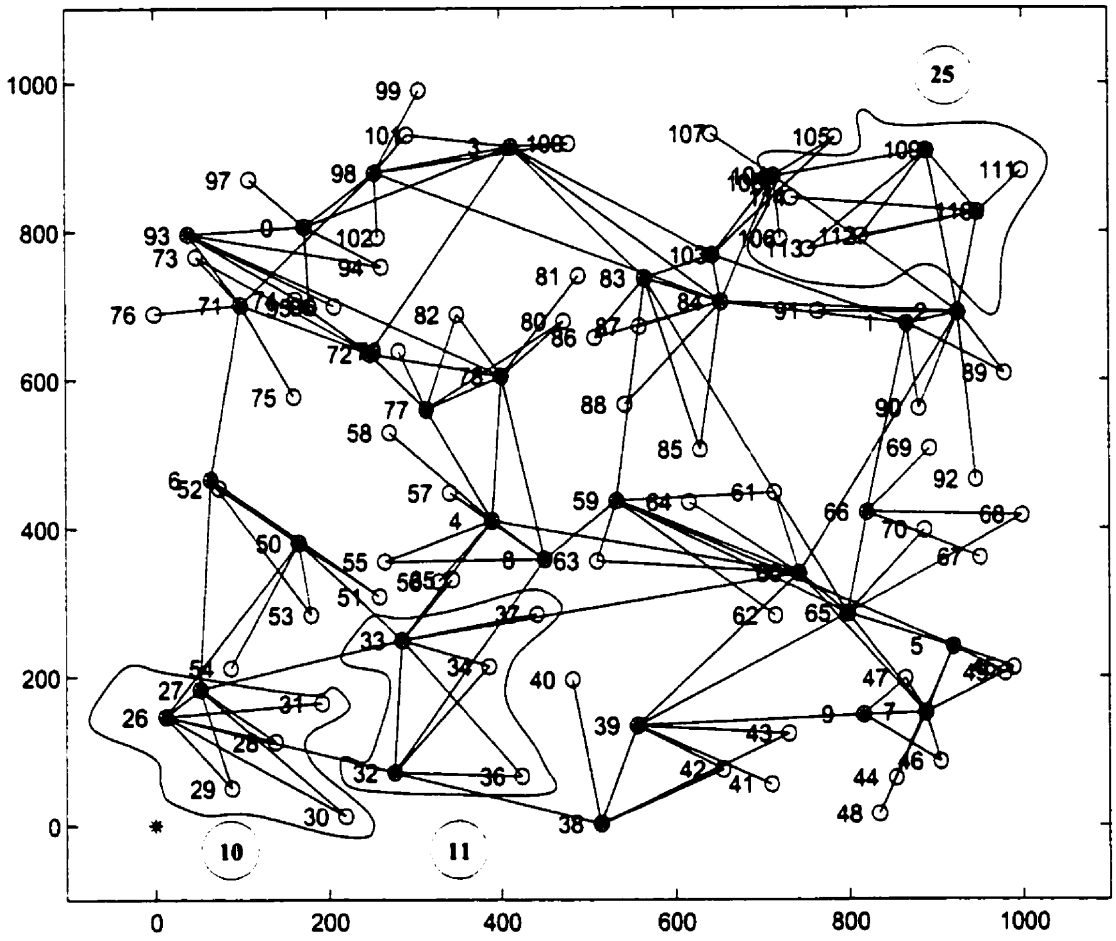


Figure 5.1: Network (1). with 16 hypernodes (nodes (10) to (25)).

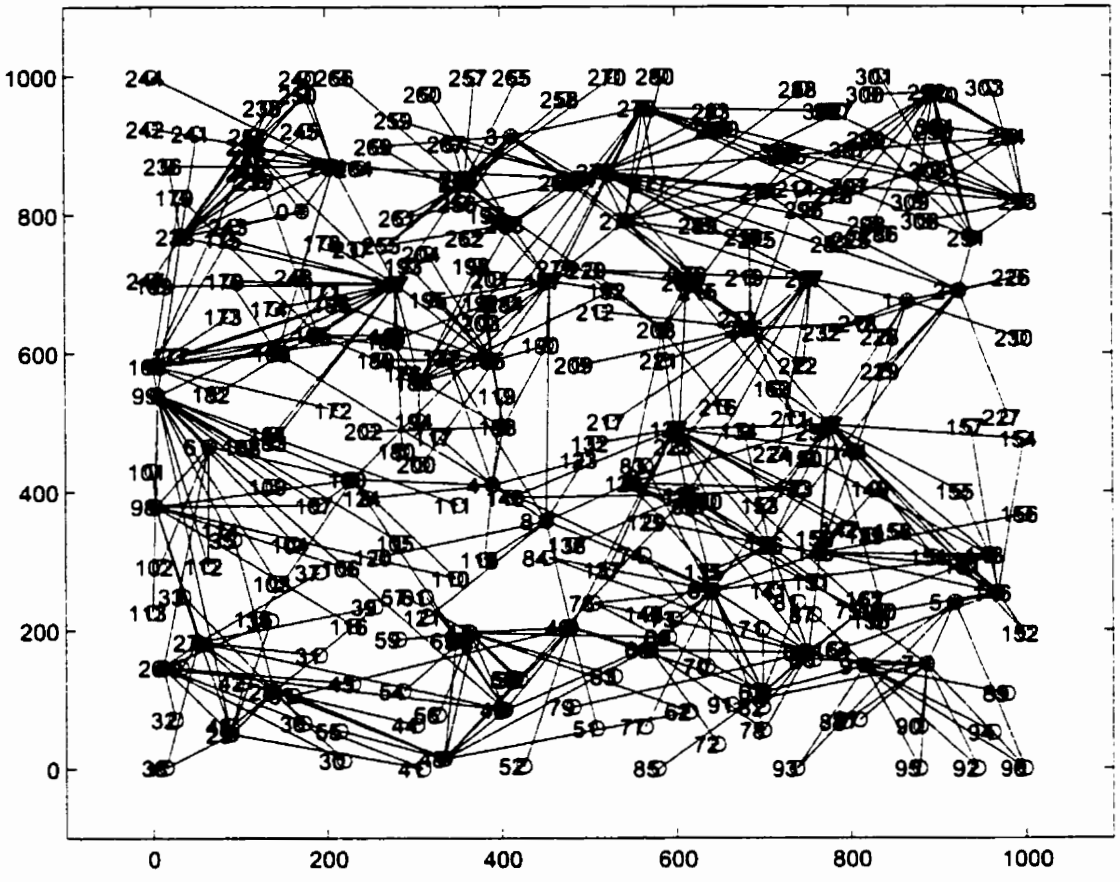


Figure 5.2: Network (2).

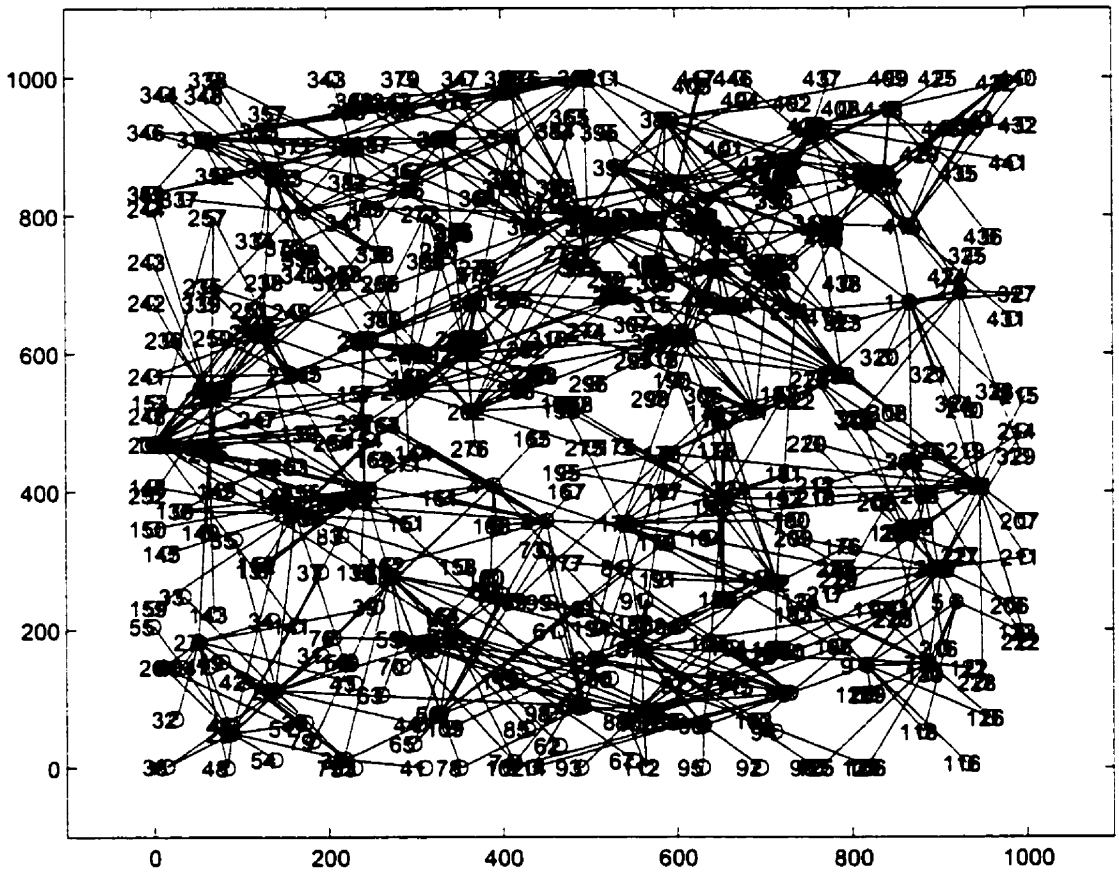


Figure 5.3: Network (3).

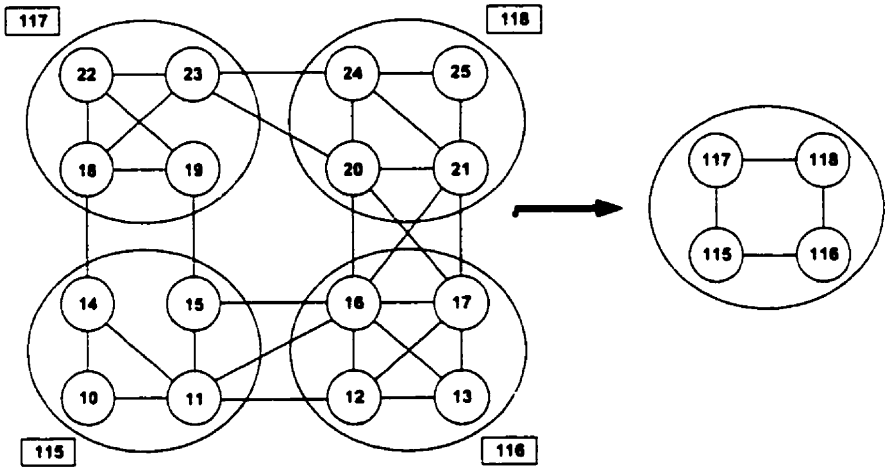


Figure 5.4: First and second layers on top of the physical network Net (1) (Figure 5.1).

Traffic Percentage	T1	T2	T3
Level (1) connections	200%	200%	200%
Level (2) connections	20%	30%	50%
Level (3) connections	10%	20%	30%

For example, with regard to Traffic Set (3), double the number of external node pairs ($2 \times N_e(N_e-1)/2$) in each layer (2) hypernode was used to generate the amount of level (1) connections. For level (2) connections, half the number of external node pairs ($0.5 \times N_e(N_e-1)/2$) in each layer (3) was used; and finally, only one third of the total number of external node pairs ($0.33 \times N_e(N_e-1)/2$) is used to generate level (3) connections.

The actual number of traffic generated for each network in each traffic set is given in Table 5.1. It is worthy to note that, due to the limitation of not more than 70% of the total incoming/outgoing fan capacity of each external node is to be exceeded, as the number of level (3) connections is increasing, the number of

level (1) connections is decreasing as can be compared by the second and third column versus the first column for each network. The reason being simply, that the higher level demands were generated first.

	Net (1)			Net (2)			Net (3)		
	T1	T2	T3	T1	T2	T3	T1	T2	T3
Level (1) connections	67	65	48	359	308	251	553	513	374
Level (2) connections	12	17	32	46	69	117	72	107	177
Level (3) connections	6	13	19	23	47	70	36	72	108
Total	85	95	99	428	424	438	661	692	659

Table 5.1: Traffic Data for Net (1), Net (2) and Net (3).

5.1.1 First Network

Tables 5.2 through 5.4 display the results for the three sets of traffic for Network (1) under different traffic load conditions. A performance comparison is carried out for the three mathematical models, namely the discrete game model (Section 3.2.3), the discrete revenue model (Section 4.4) and finally, the continuous version of the game model. For each set of traffic and for each mathematical model, three experiments were conducted with different optimization orders, namely, fair (Nash model), labeled "A" for All, then the top-down (Stackelberg model), labeled "T" and finally, the bottom-up (Stackelberg model with reversed priority), labeled "B".

In each table, the following quantities are defined:

- The number of unresolved connections. These are the connections for whom

the solver could not find a feasible point solution that guarantees their quality of service. They could be original connections, i.e. connections found in the original set of traffic loads, or inherited connections, that is, the connections that were passed-down from higher levels during previous optimization iterations.

- Their percentage. It is computed as the ratio of the unresolved connections to the total number of connections. The total number of connections is the sum of the following two quantities.

- The original number of connections obtained from the corresponding column in Table 5.1 and repeated in each table for convenience.

- The number of connections that were generated from pass-down procedures, found in the "*extra generated cons*" row in each table.

- The percentage of extra traffic size, i.e. the ratio of the number of the extra connections to their original number.

- The number of saturated or congested links, i.e. those links whose utilization factor is equal to one (100% utilization).

- The average number of links needed to route a level (i) connection.

- The total number of links used.

The comparison between the different models with different optimization orders is based on the following performance evaluation:

1. The overall throughput.

2. The link utilization distribution.

It should be noted that the throughput constitutes the more important performance measure as was discussed in Chapter 2. The VP partitioning that resulted in the benefits mentioned before and the side-effect of overall throughput reduction call for finding the balancing point at which a network should operate. This point is the one that produces the best throughput after the VP sizing operation. In these experiments, the throughput is calculated by two methods:

1. The first method expresses the throughput as the ratio between the routed connections to the total number of connections (the original set in addition to the generated connections). It is calculated using the following relation:

$$\text{Throughput} = 100\% - \text{Percentage of unresolved connections} \qquad (5.1)$$

2. The second way of computing the throughput is by actually calculating the portion of the total bandwidth demand that is routed through the network. The revenue generated is hence proportional to this quantity:

$$\mathcal{TH} = (1 - \frac{\text{Blocked requests (Mbps)}}{\text{Total bandwidth request (Mbps)}}) \times 100\% \qquad (5.2)$$

The second performance measure is simply an indication of the flow distribution in the network. The smoother the distribution is, the less the tendency of the "critical" links to become congested.

In Table 5.2, two observations can be deduced:

1. The performance of the multi-level optimization, i.e. either the top-down or the bottom-up approaches outweigh the fair (Nash) approach in all three cases of mathematical models.

2. Moreover, it can be seen from Table 5.2 that the discrete game model performs better than the other two.

	Model								
	Discrete Game			Discrete Revenue			Continuous Game		
	A	T	B	A	T	B	A	T	B
# unresolved cons	0	4	0	50	10	9	17	7	3
Percent. (unresolved)	0%	2.4%	0%	24%	5%	4.5%	9%	4%	1.6%
Original cons #	85	85	85	85	85	85	85	85	85
Extra generated cons	81	80	80	122	122	122	100	103	103
Percent. (extra)	95%	94%	94%	144%	144%	144%	118%	121%	121%
# saturated links	2	5	2	0	2	0	2	3	3
Avg # of links (1)	2	2	2	1	2	2	2	2	2
Avg # of links (2)	5	5	5	6	9	8	6	6	7
Avg # of links (3)	12	11	12	23	29	28	16	15	15
Total # of used links	180	175	178	161	183	184	180	183	186

Table 5.2: Analysis for Network (1) and Traffic set (1).

Both observations will become more evident by inspection of performance under heavier load conditions as reported in Table 5.3 and Table 5.4. Moreover, for bigger-size networks, it becomes apparent that the best technique to use for VP partitioning and routing is the top-down order of optimization using the discrete Stackelberg game model as was predicted during the design phase (Chapter 3).

	Model								
	Discrete Game			Discrete Revenue			Continuous Game		
	A	T	B	A	T	B	A	T	B
# unresolved cons	0	0	0	56	19	31	19	11	5
Percent. (unresolved)	0%	0%	0%	18%	6%	10%	7%	4%	2%
Original cons #	95	95	95	95	95	95	95	95	95
Extra generated cons	142	140	140	212	212	212	173	176	176
Percent. (extra)	150%	147%	147%	223%	223%	223%	182%	185%	185%
# saturated links	3	4	3	1	3	3	1	4	3
Avg # of links (1)	2	2	2	1	2	2	2	1	2
Avg # of links (2)	5	4	5	7	9	8	7	7	6
Avg # of links (3)	11	11	11	23	25	24	14	13	14
Total # of used links	189	188	190	169	188	189	186	190	195

Table 5.3: Analysis for Network (1) and Traffic set (2).

The throughput (calculated using Equation 5.1) for Net (1) of the three models under different traffic loads is compared using the graphical charts in Figure 5.5. On the other hand, Figure 5.6 displays the corresponding results when the throughput was calculated using Equation 5.2. As can be observed, the overall best performance

	Model								
	Discrete Game			Discrete Revenue			Continuous Game		
	A	T	B	A	T	B	A	T	B
# unresolved cons	1	0	10	80	25	43	7	12	21
Percent. (unresolved)	0.3%	0%	3%	20%	6%	11%	2%	3%	5.5%
Original cons #	99	99	99	99	99	99	99	99	99
Extra generated cons	213	213	209	299	299	299	286	282	282
Percent. (extra)	215%	215%	211%	302%	302%	302%	288%	285%	285%
# saturated links	1	2	1	1	3	1	1	1	2
Avg # of links (1)	2	2	2	1	2	2	2	2	1
Avg # of links (2)	5	5	5	7	8	7	8	6	7
Avg # of links (3)	10	10	9	20	21	20	16	13	14
Total # of used links	197	194	189	170	195	192	198	196	194

Table 5.4: Analysis for Network (1) and Traffic set (3).

is attributed to the Discrete Game Model (Figure 5.5(a)). Moreover, the multi-level optimization approaches, either the top-down or bottom-up, consistently resulted in better throughput than the single-level optimization approach. An exceptional case is the first one illustrated in Figure 5.5, in which the top-down produced less throughput than the other two. The corresponding data can be followed in the "T" columns of Table 5.2. This can be explained by the following. Under low to moderate load conditions, it is better off to use the bottom-up approach for the reason that under low load conditions, and by routing the higher priority demands first, among the multiple options of resources they can afford, they tend to choose the paths that are the only option for lower-priority demands. Otherwise, under heavier traffic loads, it is wiser to choose the top-down approach for multi-level optimization within the subnetworks.

However, when it comes to the overall links utilization, the continuous model was found to produce the smoothest distribution among the three models. Figure 5.7, gives an illustrative description of the links utilization for Network (1) and using the Top-Down approach in all three cases. This is to be expected since the capacity alloted to each VP is not restricted to take on integers values, therefore, any connection is routed over more links than in the other two discrete cases. The discrete revenue model comes in second in that regard.

Naturally, as the traffic load increases, the links utilization distribution gets smoother for all three models.

The run-time for the experiments conducted for Net (1) are shown in Figure 5.8. The following can be observed from Figure 5.8:

- In all three cases, the run-time of the "A" order of optimization is greater than the other two.

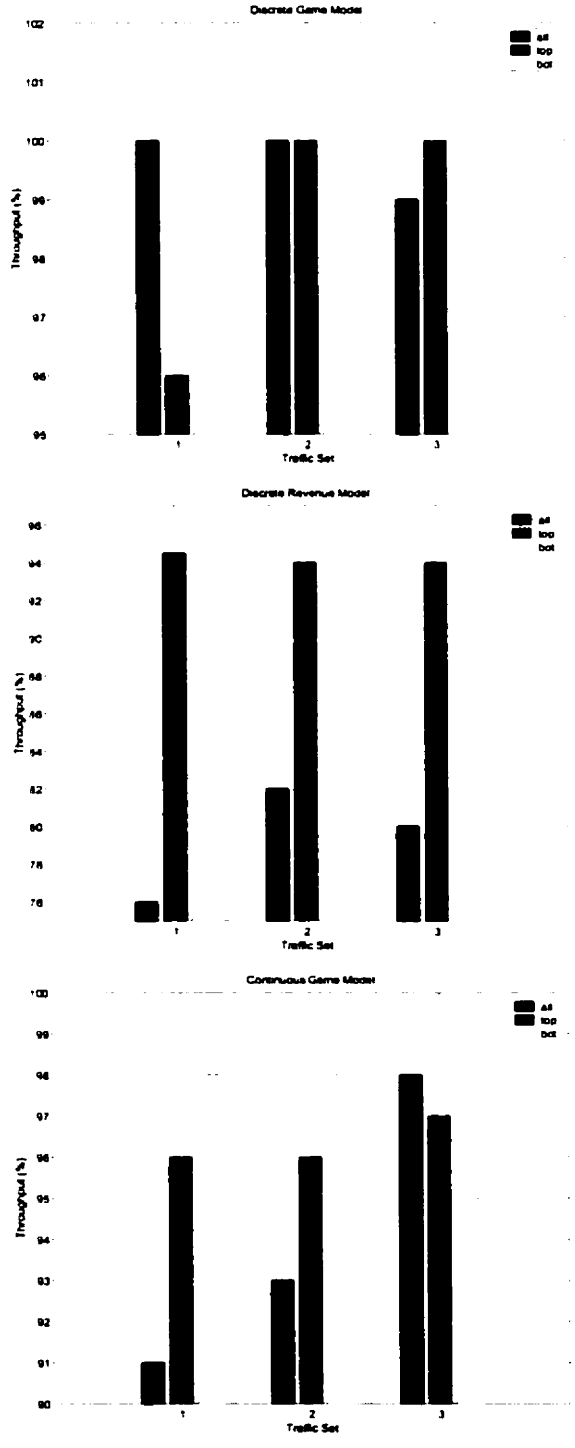
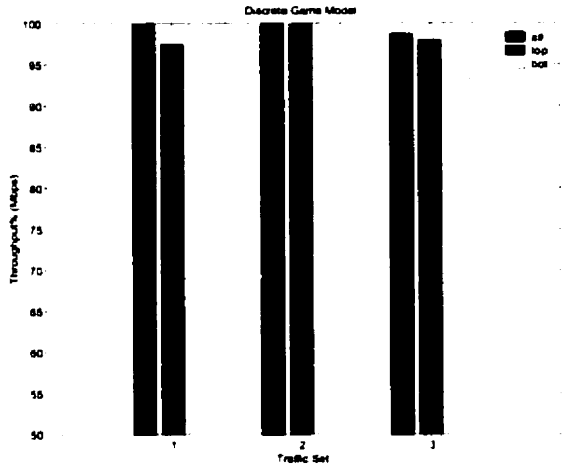
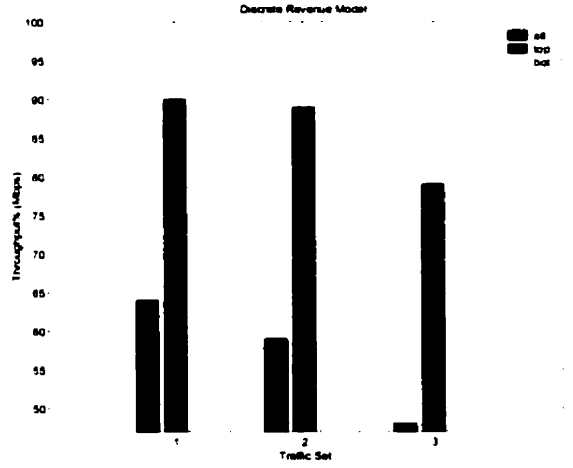


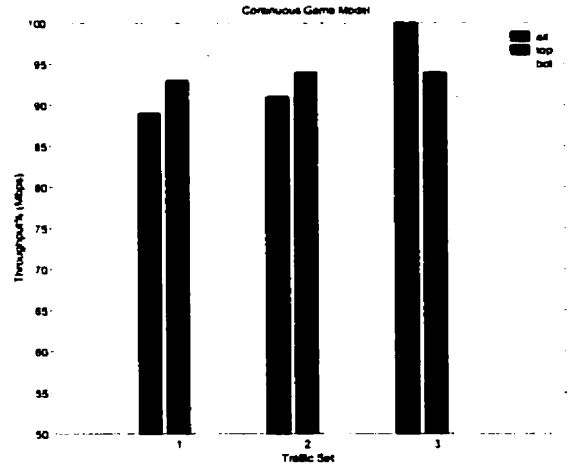
Figure 5.5: Throughput values for Net (1) (Equation 5.1) using: (a) the discrete game model, (b) the discrete revenue model and (c) the continuous game model.



(a)

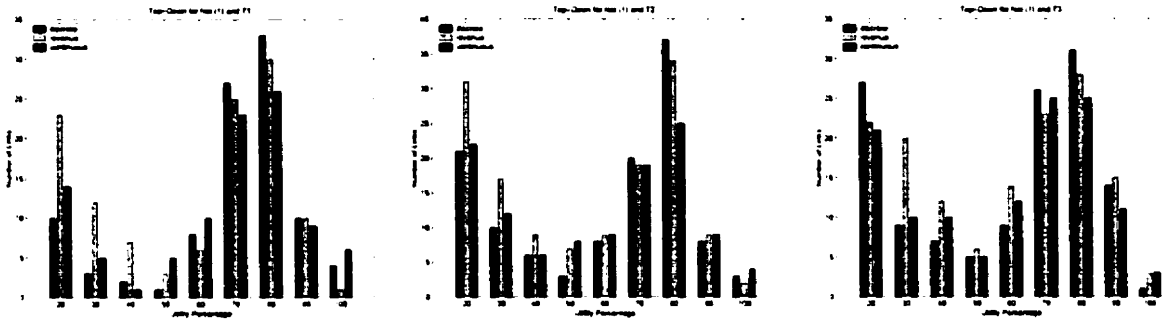


(b)



(c)

Figure 5.6: Throughput values for Net (1) (Equation 5.2) using: (a) the discrete game model, (b) the discrete revenue model and (c) the continuous game model.



(a) (b) (c)

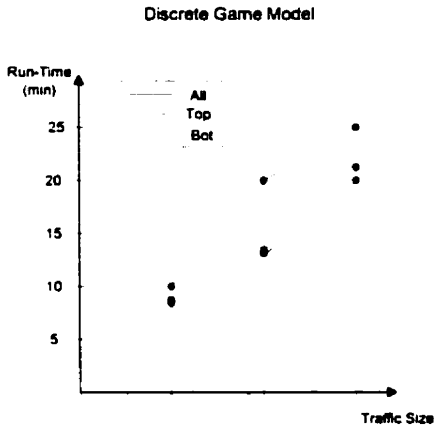
Figure 5.7: Link Utilization for Top-Down approach when used for Net (1) and under traffic loads: (a) T1. (b) T2 and (c) T3.

- The run-time of both the "T" and "B" techniques are similar more or less.
- The revenue model is much more time-consuming than the discrete and continuous game models.

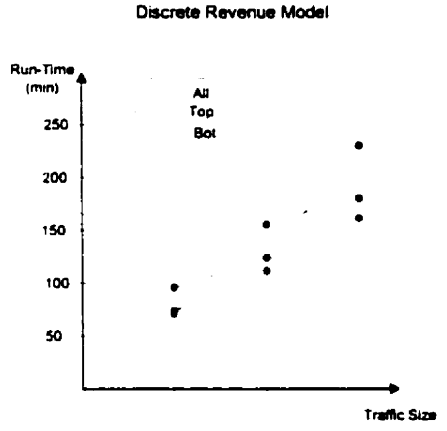
5.1.2 Second Network

For the second network. Net (2) (Figure 5.2). Tables 5.5 through 5.7 contain the results when the three mathematical models were applied to the three traffic sets. respectively. In these tables. the columns containing a N/A entry were disqualified from the link utilization comparison since the overall throughput corresponding to these experiments scored less than 50%.

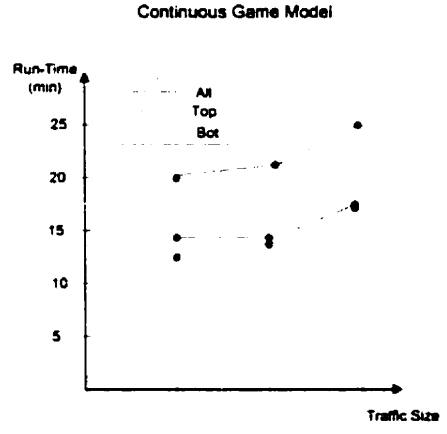
The data reported in Tables 5.5. 5.6 and 5.7 is represented graphically in the following figures. In Figure 5.9. the throughput values under different load conditions are compared for each mathematical model. As can be verified from the



(a)



(b)



(c)

Figure 5.8: Run-Time values for: (a) Discrete Game model. (b) Discrete Revenue model and (c) Continuous Game model.

	Model								
	Discrete Game			Discrete Revenue			Continuous Game		
	A	T	B	A	T	B	A	T	B
# unresolved cons	379	121	124	820	299	386	391	198	175
Percent. (unresolved)	44%	15%	15%	82%	30%	39%	36%	18%	16%
Original cons #	428	428	428	428	428	428	428	428	428
Extra generated cons	429	396	396	571	571	571	667	676	676
Percent. (extra)	100%	92%	92%	133%	133%	133%	156%	158%	158%
# saturated links	N/A	9	2	N/A	1	0	1	4	3
Avg # of links (1)		1	1		1	1	1	1	1
Avg # of links (2)		7	6		11	10	6	10	10
Avg # of links (3)		19	18		30	20	41	37	37
Total # of used links	N/A	433	448	N/A	395	401	348	412	430

Table 5.5: Analysis for Network (2) and Traffic set (1).

	Model								
	Discrete Game			Discrete Revenue			Continuous Game		
	A	T	B	A	T	B	A	T	B
# unresolved cons	527	55	97	1018	501	550	447	222	116
Percent. (unresolved)	46%	5%	9%	73%	36%	40%	27%	13%	7%
Original cons #	424	424	424	424	424	424	424	424	424
Extra generated cons	717	703	665	969	969	966	1264	1269	1269
Percent. (extra)	170%	166%	157%	230%	230%	228%	300%	300%	300%
# saturated links	0	8	0	N/A	0	0	0	1	6
Avg # of links (1)	1	2	2		1	2	1	1	2
Avg # of links (2)	4	6	7		10	7	6	9	9
Avg # of links (3)	11	18	15		18	16	40	37	35
Total # of used links	282	487	494	N/A	443	463	374	444	488

Table 5.6: Analysis for Network (2) and Traffic set (2).

	Model								
	Discrete Game			Discrete Revenue			Continuous Game		
	A	T	B	A	T	B	A	T	B
# unresolved cons	717	60	291	1329	766	953	(*)	221	
Percent. (unresolved)	49%	4%	20%	74%	42%	53%		9%	
Original cons #	438	438	438	438	438	438		438	
Extra generated cons	1037	998	1022	1361	1366	1370		1992	
Percent. (extra)	237%	228%	233%	311%	311%	313%		455%	
# saturated links	N/A	12	2	N/A	1	N/A		4	
Avg # of links (1)		2	2		1			1	
Avg # of links (2)		6	7		8			10	
Avg # of links (3)		16	10		15			39	
Total # of used links	N/A	492	487	N/A	416	N/A		478	
(*)	Experiment did not run to completion.								

Table 5.7: Analysis for Network (2) and Traffic set (3).

figure, the "Top-Down" order of optimization faired better that the "All" approach and better or equal to the "Botton-Up" approach in both discrete models. In the continuous case, however, the "Bottom-Up" approach was found to be better or equal to the Top-Down approach.

If the throughput comparison is based on the Top-Down approach only between the three mathematical models, it can be observed, from Figure 5.11, that the discrete game model is consistently superior to the other two models under any load condition.

As for Net (1), the links utilization are compared and Figure 5.12 outlines this comparison. As in the previous scenario, the continuous model offered a smoother distribution than the discrete game model. The discrete revenue model values can be discarded since some of them resulted in less than 50% throughput. Therefore they do not reflect the real figure of the total number of links utilized nor the utilization distribution, for that matter.

The run-time for the experiments conducted for Net (2) are shown in Figure 5.13. Due to the fact that the "A" experiment in the continuous case under traffic load T3 did not run to completion, only the run-time data for the discrete models is shown in the figure. As is observed from Figure 5.13 (b), the sequential operation of the proposed system under the discrete revenue model is prohibitevely time-consuming, whereas, for the same problem size, the discrete game model exhibited, not only better throughput, but also a considerably faster operation, especially when using the multi-level optimization approach versus the single-level counterpart.

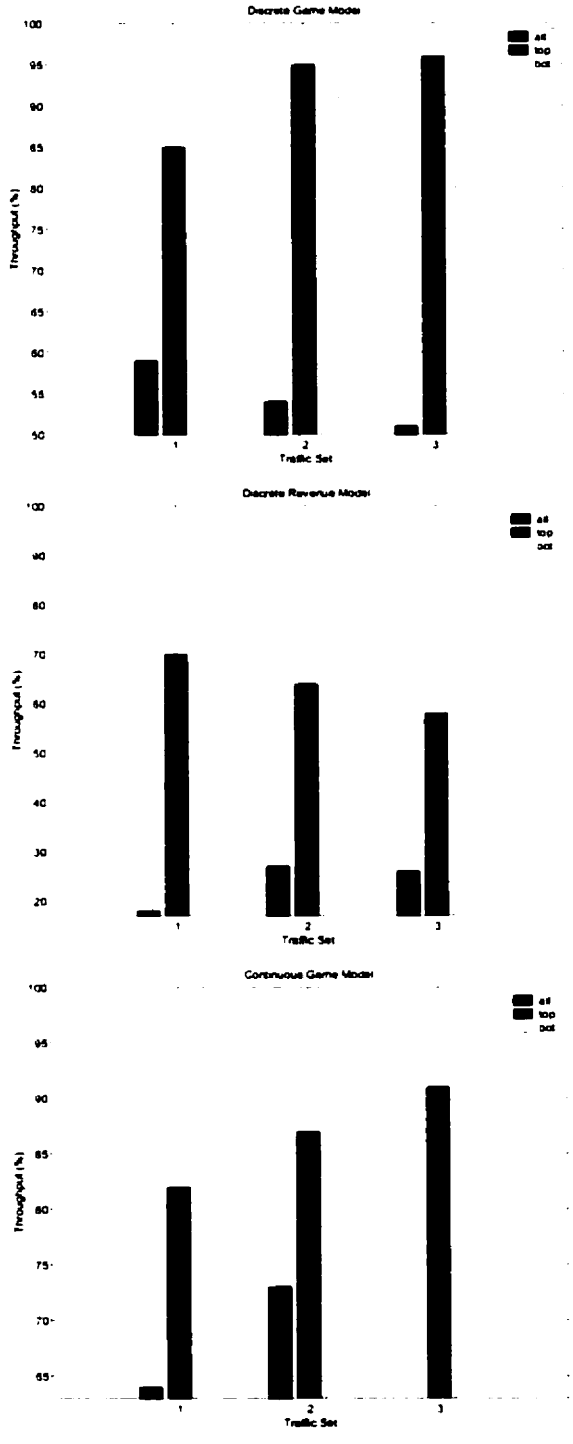


Figure 5.9: Throughput values (Equation 5.1) for Net (2) using: (a) the discrete game model, (b) the discrete revenue model and (c) the continuous game model.

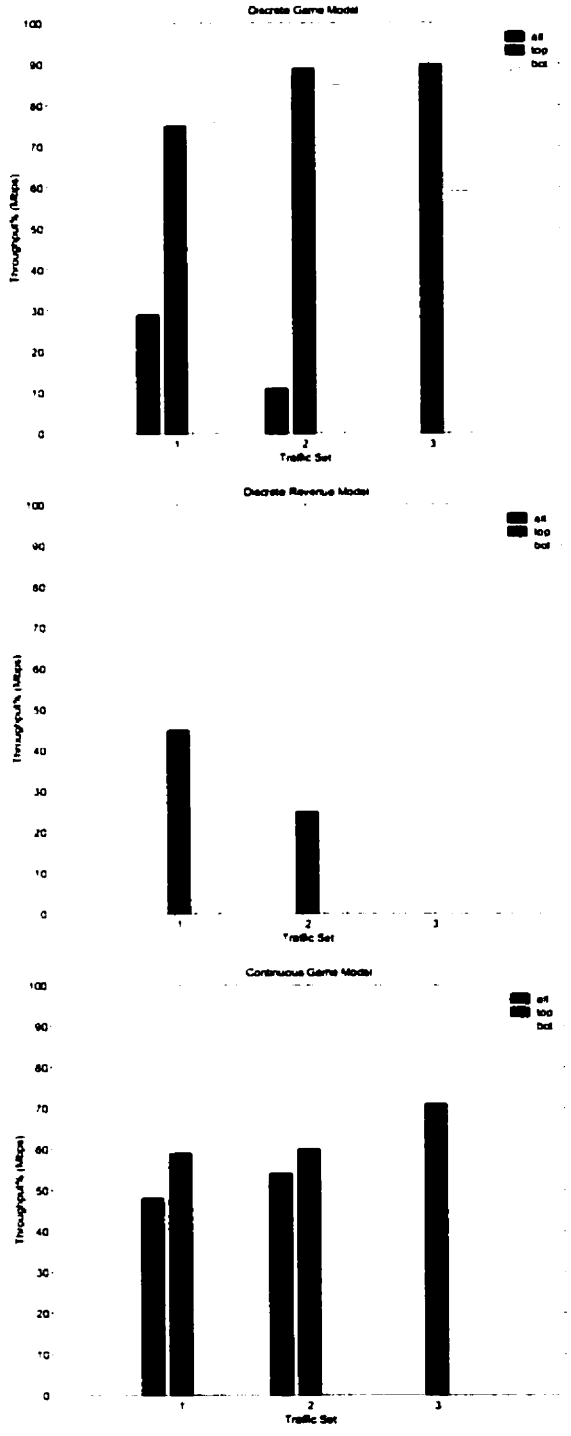


Figure 5.10: Throughput values (Equation 5.2) for Net (2) using: (a) the discrete game model. (b) the discrete revenue model and (c) the continuous game model.

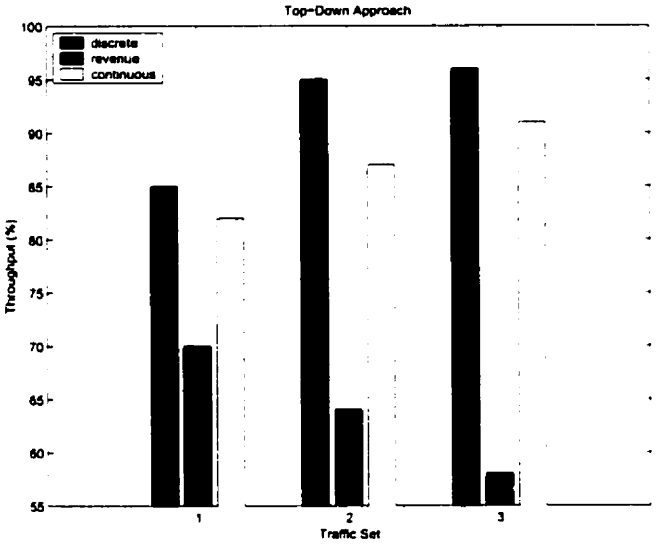
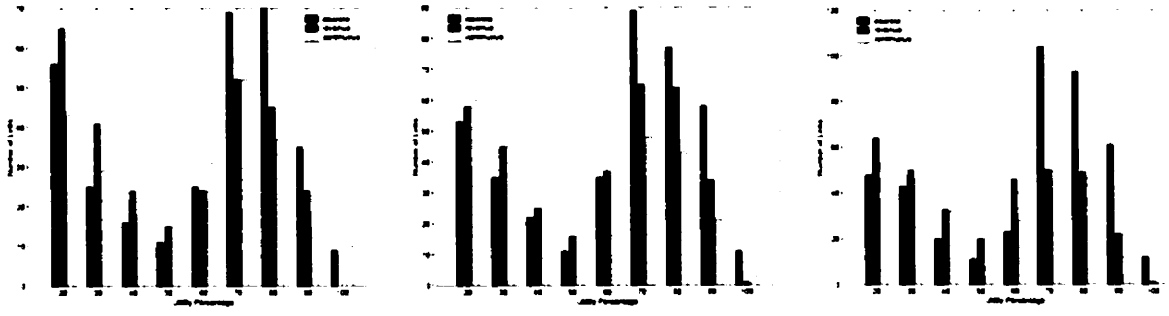
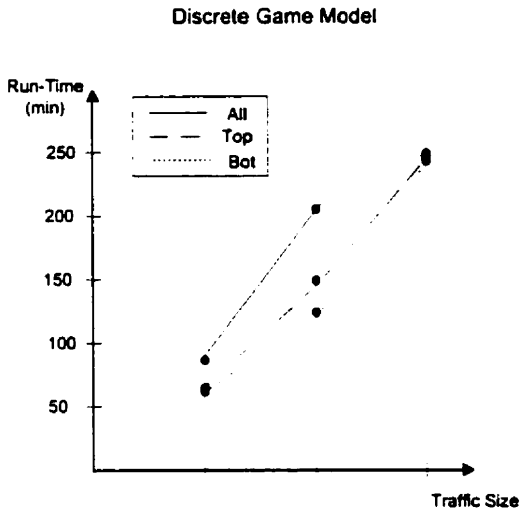


Figure 5.11: Throughput values for Net (2) when using the Top-Down Approach and under increasing traffic demands.

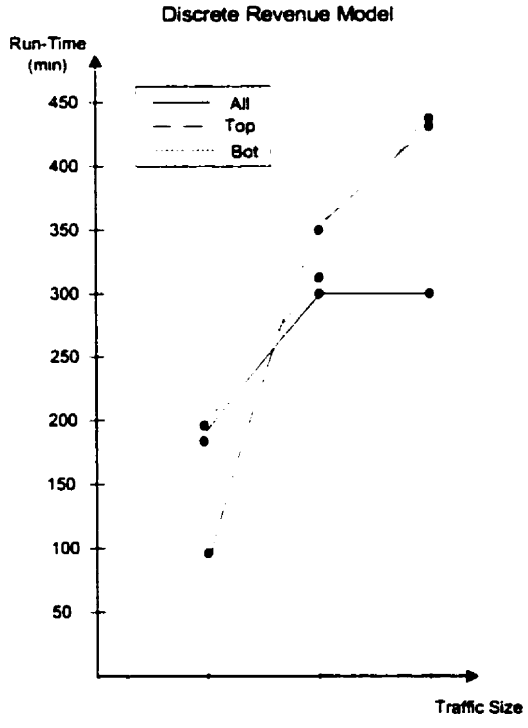


(a) (b) (c)

Figure 5.12: Link Utilization for Top-Down approach when used for Net (2) and under traffic loads: (a) T1. (b) T2 and (c) T3.



(a)



(b)

Figure 5.13: Run-Time values for: (a) Discrete Game model and (b) Discrete Revenue model.

5.1.3 Third Network

Finally, for the largest network, Net (3), based on the experimental data of the previous two networks, only the discrete game model was considered and used in the experiments shown in Table 5.8.

Table 5.8 demonstrates the results obtained when comparing the three different orders of optimization. It can be noticed that the results for the "A" approach and the traffic set T3 are not reported. This is due to the problem size: a run-time error of insufficient memory resulted in each time this experiment was conducted. Once again, it can be observed that both the "T" and "B" approaches are equally

	T1			T2			T3		
	A	T	B	A	T	B	A	T	B
# unresolved cons	959	403	331	1272	564	576		925	997
Percent. (unresolved)	73%	32%	26%	70%	32%	32%		40%	43%
Original cons #	661	661	661	692	692	692		659	659
Extra generated cons	655	589	590	1118	1061	1080		1675	1668
Percent. (extra)	99%	89%	89%	162%	153%	156%		254%	253%
# saturated links	0	1	4	N/A	0	2		12	4
Avg # of links (1)	1	1	1		1	1		1	2
Avg # of links (2)	3	7	7		6	6		6	6
Avg # of links (3)	7	19	18		17	12		11	7
Total # of used links	207	534	583	N/A	535			589	596

Table 5.8: Analysis for Network (3).

good in terms of throughput, with the "B" being better in the low traffic loads and the "T", on the other hand, being better in the heavy traffic loads.

It can be observed from Figure 5.14 that the overall performance in terms of throughput becomes significantly poor with the increased network size.

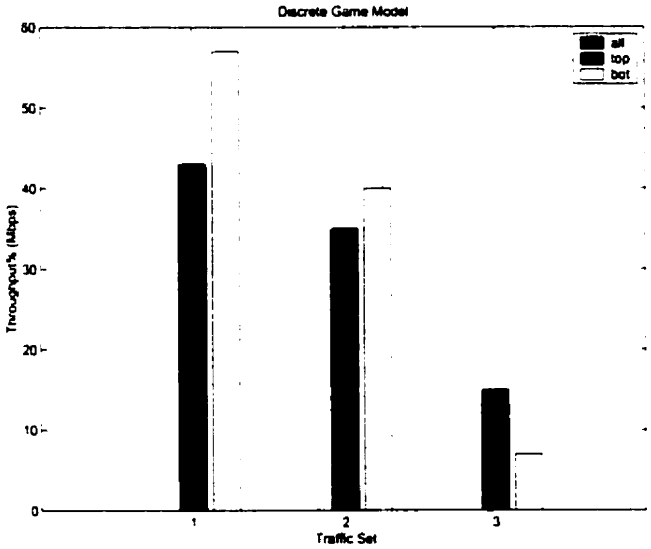


Figure 5.14: Throughput values for Net (3) (Equation 5.2)

5.2 Summary of Results

A summary of the experimental results of the previous section can be stated as follows:

- Under any traffic load and using any of the three described mathematical models, the multi-level optimization was found to exhibit consistently a better

performance in terms of higher throughput and lower run-time values than the single-level optimization.

- Among the three mathematical models, namely, the discrete game model, the discrete revenue model and the continuous game model, the first model was found to be superior to the other two.

- If the single-level approach is discarded from the comparison, then it can be deduced from the previous analysis that:

  - Under low traffic loads, the "Bottom-Up" approach behaves better than the "Top-Down" approach for the reasons discussed previously.

  - As the traffic size increases, the "Top-Down" approach performs better than the "Bottom-Up".

  Therefore, when solving the VP sizing problem of a network under moderate to high traffic conditions, it is recommended to use the discrete game model with a "Top-Down" order of optimization.

Chapter 6

Summary

In the present work, we addressed the problem of virtual path allocation in large-scale ATM networks. For a hierarchically structured network, the presented system described a distributed algorithmic framework for the coordination of network management and traffic control. The capacity allocation problem within a subnetwork was modeled as a noncooperative game, where virtual circuits, or large portions of bandwidth demand requested by service providers, were competing over links utilization. The operating point for each subnetwork was selected to be the equilibrium point that resulted in the best performance and highest overall network throughput. In the sequel, a summary of the characteristic points in the current system is presented.

6.1 Summary

- In loss networks, the *Erlang loss formula* $\mathcal{E}(\rho, C)$ gives the blocking probability in a link comprising C circuits when the offered traffic load is ρ connections

per unit time [10, 36]. The partition function or normalization constant in this formula is computationally challenging. In order to overcome this computational complexity, approximation methods for the computation of the normalization constant were introduced, such as the *Erlang fixed point* approximation [68].

- The use of virtual paths in ATM networks reduces the call set-up delays, simplifies the hardware in the transit nodes and provides simple virtual circuit admission control. However, it also reduces the degree of capacity sharing and, thus, increases the overall call blocking rate. Since the VP allocation problem is NP-hard, the solution to the optimization problem is therefore based on heuristics. Work found in the literature in this domain optimizes a cost function, such as decreasing the total call blocking rate or increasing the network revenue, subject to capacity and degree of service constraints.

- One way to address the VP allocation problem is based on a game theoretic approach. We used the noncooperative game model at the basis of the VP capacity allocation problem within each peer group. The Stackelberg model acts more in favour of the leaders than a Nash game. Therefore, providing lower blocking probabilities for demands in higher levels.

- A hierarchical organization of large-scale networks was presented in which the physical layer nodes are clustered into conceptual hypernodes in multiple layers. Three-layer networks were used to demonstrate the validity of the present technique in VP partitioning and routing.

- Information passed in the network is only in a vertical direction and not among peer groups. In other words, information concerning links states and topology of the lowest hierarchical level is encoded and sent upward to the

parent node in the following level, while traffic demands are passed downward from a node to its children. There is no interaction between *siblings*, so to speak. This last feature make the current system lend itself naturally to a distributed computation paradigm in which network managers existing in the same level can process their data in parallel.

- Moving upward in the hierarchy, the topology and link state information were propagated using an abstraction of the links available bandwidth as was explained in Section 3.2.1. Links connecting peer groups were aggregated in a single link whose bandwidth is the sum of that of its constituents. Total flow that can pass through a hypernode in a given direction was obtained using the Maximum-Flow principle in that direction.

- The implementation of the current system was done using the object-oriented language C++ and the Tomlab [27] optimization solver. The type of interaction between the different modules in the system was described and a description of the data structure and program flow was provided.

6.2 Summary of Contributions

- At the time when the current system was initially proposed, one could hardly find in the literature any work dealing with large-scale networks. The survey done then showed that the VP partitioning and/or routing problem was solved for small to medium size networks not exceeding 40 nodes in total. Our work, on the other hand, deals with networks in the range of 100 to 500 nodes in size. This constitutes the first contribution of this work.

- Due to the difficulty of acquiring real-world data of large-scale networks and

their traffic patterns, rose the need to develop, from scratch, a random topology generator and a random traffic generator (Section 4.1.1). The generated networks and traffic demands formed the test-beds used for the current system evaluation, and they also constitute a second contribution.

- The clustering technique, though is not new and is following the guidelines for the PNNI networks [17], is implemented in such a way that can be generalized to the use for any given network and is not limited to the ones generated by our topology generator module. Hence, creating a useful tool for the conceptual organization of networks in hierarchical layers.

- The hierarchical VP partitioning and routing technique itself, with the inter-layers protocol and its multi-level optimization nature is the main contribution of this work. More precisely, the *Update-Status* routine that solves the routing details in each subnetwork recursively, as explained in Sections 4.2.2 and 4.3, is a new addition to the available bandwidth allocation and routing techniques existing in the telecommunication field today.

6.3 System Evaluation and Potential Extensions

The proposed approach for VP capacity allocation in large-scale networks is based on network segmentation into smaller subnetworks and providing for coordination between the network controllers. In the sequel, we discuss some features associated with the presented system.

- As was mentioned before, the benefits of a hierarchical structure are
  - Avoiding the combinatorial complexity inherent in the original network topology.

  - The segmentation and abstraction of the global network into smaller components leads to added flexibility in terms of scalability. In other words, accounting for future network modification, adding or reducing the number of switching nodes in the network affects only the topology of the peer group (and its parent) where this change has occurred. It can also account for faulty or saturated links or for switches that are temporarily out of service.
  - Though in the present system, the optimization process is consistent throughout the different layers, it need not be so, and each local manager is free to select its own mathematical model and force its own set of constraints, as it best suits the operation in its administrative domain.

- However, an obvious weakness in this approach is the reduction in accuracy due to the loss of information resulting from the introduced abstraction. When a subnetwork topology is aggregated, it does not in reality represent the total capacity that can be shared. Consequently, the VP dimensioning of the upper logical node is restricted to the provided capacity information.

- In the presented approach, we discarded the effect of propagation delays in the network links. One possible extension to the developed system could be that of incorporating another information aggregation representing the delays. The constraints then can include some delay bounds.

- Another possible extension is to study other clustering techniques instead of the one implemented in this system. These techniques can use different criteria for clustering the nodes pertaining to the same peer-group [4, 5].

- Instead of the sequential operation, the parallel processing of hypernodes in

the same hierarchical level is favoured over the current implementation and is recommended for consideration for future work.

- As was mentioned in the course of the system description, a more greedy approach can be used instead of the current fair approach in relation to the bandwidth partitioning among children edges of a given hyperedge. This alternative is a primary candidate for study as a potential extension to the present system.

- Some of the challenges encountered during the development phase include, but are not limited to:

  - The selection of the proper optimization solver. Even when we decided to interface with the Tomlab library, a remodeling of the problem formulation was a necessity in order to comply with the memory constraints imposed by Matlab. The modification of the mathematical model to reduce the problem size is given in Appendix B.

  - In order to check for the soundness of the VP partitioning and routing algorithm, the debugging and tracking in such large networks was not trivial to say the least. Other than being the major time-consuming factor in the present work, extra care intended to make the route selection as fair as possible.

  - The recursive nature of the *Update-Status* was particularly hard to implement (Sections 4.2.2, 4.3). It had to be *(i)* as general as possible to suit any network topology and at the same time *(ii)* deal with the detailed topology of the network, and its subnetwork, under consideration.

- Based on the analytical discussion provided in the previous chapter, the dis-

crete game model outweighs the classical one and its own continuous version in performance. This is the recommended approach to solve the VP sizing problem in large ATM networks.

- As was discussed in Chapter 3, no closed-form formulation could be obtained that would express the way the network manager can adjust the individual payoff functions of the self-optimizing users due to the complexity of the large-size problems. Therefore, simulation techniques were used in order to assess the validity of the implemented model.

  One of the main issues that needs to be addressed in subsequent research is the effect of cost perturbation on both the overall network performance and the users behaviour. Namely, the quantity α_i in Equation 3.2 (repeated here for convenience)

$$F_i^l(c_i^l, C^l) = c_i^l \cdot \frac{\alpha_i}{1 - C^l/B^l} \qquad (3.2)$$

  should be expressed as a function of the network manager strategy; for instance:

$$\alpha_i = f_i(C^o, \mathbf{C})$$

  where C^o is the strategy profile of the network manager imposed on the users to drive the network's performance to a desirable operating point, and $\mathbf{C}$ is the strategy profile of the users. The global network objective can then be expressed by maximizing a certain gain function:

$$\text{Max} \qquad \text{Rev}(\overline{\alpha}, \mathbf{C})$$

  where $\overline{\alpha}$ is the set of costs assigned to the users. Since expressing the functions f_i, and Rev in closed-form notation is intractable, the future work should

deduce the empirical format of these functions and study the sensitivity of this gain function to perturbations in the value of $\overline{\alpha}$.

- The presented approach is not limited to the application with respect to ATM networks only, but also for the Multi-Protocol Label Switching based networks (MPLS) [18, 16, 33]. Therefore, a proposed extension to the presented system is to suitably tailor it for the network management and establishment of guaranteed bandwidth tunnels in MPLS settings.

Appendix A

Classes Implementation

A.1 Network Data Structures

A.1.1 Node Class

```
/*************************************************************************/
/*              NODE / GRAPH (hypernode) CLASS                           */
/*************************************************************************/
class cNode
{
  private:

  int      id;                  // node id
  int      loc;                 // node location in V
  int      level;               // level in hierarchy structure
  int      degree;              // used for undirected graphs
  boolean  border;              // TRUE for a border-node
  nodeType type;                // external = edge node,
                                // internal = switch node

  double   status;              // current capacity
  double   capacity;            // maximum capacity

  point2d  coord;               // physical map location

  node         parent;          // parent graph
  TcList<edge> adj_edges;       // list of adjacent edges
```

```
/*-----------------------------------------------------------------*/

// the following lists are NULL for the physical layer (level = 1)

TcList<node>  V;              // children/subgraph nodes
TcList<edge>  E;              // children/subgraph edges
TcList<node>  B;              // border-nodes in level beneath
TcList<edge>  L;              // logical links connecting those
                              // border-nodes

/*-----------------------------------------------------------------*/
public:

cNode();
cNode(const cNode& N);
~cNode();

void     Copy(const cNode& N);
cNode&   operator=(const cNode& N);
boolean  operator==(const cNode& N) const;
boolean  operator!=(const cNode& N) const;

/*-----------------------------------------------------------------*/

int      Id() const               { return( id ); }
void     Id(int newid)            { id = newid; }
int      Loc() const              { return( loc ); }
void     Loc(int newloc)          { loc = newloc; }
int      Level() const            { return( level ); }
void     Level(int newlevel)      { level = newlevel; }
int      Degree() const           { return( degree ); }
void     Degree(int newdegree)    { degree = newdegree; }
boolean  Border() const           { return border; }
void     Border(boolean value)    { border = value; }
nodeType Type()  const            { return( type ); }
void     Type( nodeType newtype ) { type = newtype; }

double  Status() const            { return( status ); }
void    Status(double newstatus)  { status = newstatus; }
double  Capacity() const          { return( capacity ); }
void    Capacity(double newcap)   { capacity = newcap; }
point2d Coord() const             { return( coord ); }
```

```
  void    Coord(point2d newcoord)    { coord = newcoord; }
  void    Coord(double x, double y)  { coord.x = x; coord.y = y; }

  node    Parent() const             { return( parent ); }
  void    Parent(node newparent)     { parent = newparent; }

  TcList<edge>  Adj_Edges() const    { return( adj_edges ); }
  void          Adj_Nodes( TcList<node>& N );
  boolean       Add_Adj_Edge( edge e );
  boolean       Delete_Adj_Edge( edge e );

  /*-----------------------------------------------------------------*/

  int   Node_Num() const { return ( V.Size() ); }
  int   Edge_Num() const { return ( E.Size() ); }

  TcList<node>  V_Nodes()   const { return ( V ); }
  TcList<edge>  E_Edges()   const { return ( E ); }
  TcList<node>  B_Nodes()   const { return ( B ); }
  TcList<edge>  L_Edges()   const { return ( L ); }

  boolean   Add_V_Node( node n );
  boolean   Delete_Node( node n );
  boolean   Add_E_Edge( edge e );
  boolean   Delete_Edge( edge e );
  boolean   Add_B_Node( node n );
  boolean   Add_L_Edge( edge e );

  node  First_Node()   const     { return ( V[0] ); }
  node  Last_Node()    const     { return ( V[V.Size()-1] ); }
  node  Pred_Node(node v)  const;
  node  Succ_Node(node v)  const;

  edge  First_Edge()    const    { return ( E[0] ); }
  edge  Last_Edge()     const    { return ( E[E.Size()-1] ); }
  edge  Pred_Edge(edge e)  const;
  edge  Succ_Edge(edge e)  const;

  boolean  Save(FILE* fp);
  boolean  Load(FILE* fp, node* n_add, edge* e_add);

};
```

A.1.2 Edge Class

```
/*************************************************************************/
/*              EDGE  (hyperedge) CLASS                                  */
/*************************************************************************/
class cEdge
{
  private:

  int           id;                 // edge id
  int           loc;                // index in |E|
  int           level;              // level in hierarchy structure
  node          s;                  // source node
  node          t;                  // target node
  edgeType      type;
  double        capacity;           // maximum capacity
  double        status;             // current capacity

  graph         parent;             // parent graph
  TcList<edge>  children;           // constituent edges beneath

  public:

  cEdge();
  cEdge(const cEdge& E);
  ~cEdge();

  void     Copy(const cEdge& E);
  cEdge&   operator=(const cEdge& E);
  boolean  operator==(const cEdge& E) const;
  boolean  operator!=(const cEdge& E) const;

  int      Id() const             { return( id ); }
  void     Id(int newid)          { id = newid; }
  int      Loc() const            { return( loc ); }
  void     Loc(int newloc)        { loc = newloc; }
  int      Level() const          { return( level ); }
  void     Level(int newlevel)    { level = newlevel; }
  node     S()  const             { return( s ); }
  void     S(node source)         { s = source; }
  node     T()  const             { return( t ); }
  void     T(node target)         { t = target; }
```

```
  edgeType Type()  const              { return( type ); }
  void     Type( edgeType newtype )  { type = newtype; }

  double  Status() const            { return( status ); }
  void    Status(double newstatus) { status = newstatus; }
  double  Capacity() const          { return( capacity ); }
  void    Capacity(double newcap)  { capacity = newcap; }

  node          Parent() const         { return( parent ); }
  void          Parent(node newparent) { parent = newparent; }
  TcList<edge>  Children() const       { return( children ); }

  void     cEdge::Great_Grand_Children( int desired_level,
                                        TcList<edge>& grand_children );
  void          Add_Child( edge e );

  edge  First_Edge()  const     { return ( children[0] ); }
  edge  Last_Edge()   const
                 { return ( children[ (children.Size()) - 1] ); }
  edge  Pred_Edge(edge e)  const;
  edge  Succ_Edge(edge e)  const;

  boolean  Save(FILE* fp);
  boolean  Load(FILE* fp, node* n_add, edge* e_add);
};
/*****************************************************************************/
```

A.2 Traffic Data Structure

```
/*************************************************************************/
/*                    CONNECTION CLASS DEFINITION                        */
/*************************************************************************/

class cConnection
{
  private:

  int      id;
  // source node originating this connection
  node     original_source;
  // destination node for this connection
  node     original_destination;
  // source node in the current optimization iteration
  node     source;
  // destination node in the current optimization iteration
  node     destination;
  double   bw_required;   // bandwidth requirement (Ci)
  int      level;         // connection belonging to which level in hierarchy
  double   revenue_rate;  // revenue in $/unit of time generated by this conn
  double   arrival_rate;     // arrival rate (lambda)
  double   qos;              // quality of service required

  conptr          parent;    // parent connection
  // route num to which this child connection belongs
  int             route_id;
  // children connections forked-out from this one
  TcList<conptr>  children;
  // set of routes on which this connection is routed
  TcList<route>   Routes;
  TcList<n_route> Node_Routes;

  TcList<listdoublelistptr>  Routes_Bw;

  public:

  cConnection();
  cConnection(const cConnection& Con);
  cConnection(node s, node d);
```

```
~cConnection();

void          Copy(const cConnection& Con);
cConnection&  operator=(const cConnection& Con);
boolean       operator==(const cConnection& Con) const;
boolean       operator!=(const cConnection& Con) const;

int    Id()                 const    { return( id ); }
void   Id( int newid )               { id = newid; }
node   Orig_Source()        const    { return( original_source ); }
void   Orig_Source(node s)           { original_source = s; }
node   Orig_Destination()   const    { return( original_destination ); }
void   Orig_Destination(node d)      { original_destination = d; }
node   Source()             const    { return( source ); }
void   Source(node s)                { source = s; }
node   Destination()        const    { return( destination ); }
void   Destination(node d)           { destination = d; }
double BW()                 const    { return( bw_required ); }
void   BW(double bw)                 { bw_required = bw; }
int    Level()              const    { return( level ); }
void   Level(int lev)                { level = lev; }
double Revenue_Rate()       const    { return( revenue_rate ); }
void   Revenue_Rate(double r)        { revenue_rate = r; }
double Arrival_Rate()       const    { return( arrival_rate ); }
void   Arrival_Rate(double a)        { arrival_rate = a; }
double QoS()                const    { return( qos ); }
void   QoS(double q)                 { qos = q; }

conptr          Parent()       const    { return( parent ); }
void            Parent( conptr npar )   { parent = npar; }
int             Route_Id()     const    { return( route_id ); }
void            Route_Id( int newid )   { route_id = newid; }
TcList<conptr>  Children()     const    { return( children ); }
void            Children( const TcList<conptr>& NewList )
{ children =  NewList; }
TcList<route>   R_Routes()    const  { return( Routes ); }
void            R_Routes( const TcList<route>& NewRoutes )
{ Routes = NewRoutes; }
TcList<n_route> NR_Routes()   const  { return( Node_Routes ); }
void            NR_Routes( const TcList<n_route>& NewRoutes )
{ Node_Routes = NewRoutes; }
```

```
  TcList<listdoublelistptr>    R_BW()   const  { return( Routes_Bw ); }
  void     R_BW( const TcList<listdoublelistptr>& NewR_BW )
  { Routes_Bw = NewR_BW; }

  void   Add_Child( conptr child, int r_id );
  void   Remove_Child( conptr child );
  void   Remove_Child( int child_id );
  void   Add_Route( route R, n_route NR, double bw );
  void   Copy_Routes( const cConnection& Con );
  int    Print_Connection( FILE* fp );

  boolean  Save( FILE* fp );
  boolean  Load( FILE* fp, conptr* con_add, node* n_add, edge* e_add );

};   // end class cConnection
/*************************************************************************/
```

Appendix B

Examples of Matlab Files

In this appendix, a description of the Matlab file that is used as an input to the Tomlab solver is given in Section B.1. The Matlab files implementing the objective functions for the game model and the classical revenue model are outlined in Section B.2.

B.1 M-File used for Network Modeling

A sample Matlab file (.m file) containing the mathematical model for a network is given below. The network example shown has 4 nodes, $I = 6$ connections and $R = 12$ potential routes. The parameters appearing in the m-file have the following interpretation:

- The constants: connection_num, link_num and route_num are the number of connections, links and routes, respectively. Let's denote set of connections by I, the set of links by L and the set of routes by R; with dimensions n_I, n_L and n_R, respectively.

- Lambda_i, Alpha_i, Ci and QoS_i are the arrival rate, cost/revenue, bandwidth requirement and quality of service for connection i, respectively.
- BL vector ($n_L \times 1$). The l^{th} entry denotes the maximum capacity of link l.
- IR matrix ($n_I \times n_R$). Each row in this matrix corresponds to one connection: and each column corresponds to one route. If $IR(i,r) = 1$, then route r is one of the set of routes that connection i can use.
- RL matrix ($n_R \times n_L$). The $(r,l)^{th}$ is equal to one if link l is part of route r.
- Y vector ($n_R \times 1$). This is the independent variables vector. The n_R unknowns that we need to solve for are the assigned bandwidth values to each route $r \in R$.
- Instead of solving a problem of size n_R, a simple modification was introduced [28] to reduce the problem size. By inspection of the above formulation, the following relation is true:

$$IR \times Y = Ci$$

For instance, the first three rows in IR can be written as:

$$y_1 + y_2 = 7$$
$$y_3 + y_4 = 2$$
$$y_5 + y_6 = 7$$
$$\vdots$$

We can eliminate n_I variables by noting that:

$$y_2 = 7 - y_1$$
$$y_4 = 2 - y_3$$
$$y_6 = 7 - y_5$$
$$\vdots$$

Then we can denote the new set of variables X of size that is equal to $(n_R - n_I)$. and that can be expressed as follows:

$$x_1 = y_1 \quad or \quad y_1 = x_1$$
$$x_2 = y_3 \quad or \quad y_3 = x_2$$
$$x_3 = y_5 \quad or \quad y_5 = x_3$$

and

$$y_2 = 7 - x_1$$
$$y_4 = 2 - x_2$$
$$y_6 = 7 - x_3$$

Hence. the following transformation from the set X of variables to the Y set of variables takes place:

$$Y = YX \times X + CCi$$

where:

- YX matrix $(n_R \times (n_R - n_I))$ is the transformation matrix.
- CCi vector $(n_R \times 1)$ that can be obtained by reordering the y variables in the example above.
- IRnew matrix $(n_I \ times (n_R - n_I))$. The new IR matrix corresponding to the new set X of variables.
- In case of continuous variables. the following line

```
IntVars = [1:(route_num - connection_num)];
```

is modified to:

```
IntVars = [];
```

Contents of "net_119_2.m"

```
function net_119_2 (output_file)

global connection_num
global link_num
global route_num
global Lambda_i
global Alpha_i
global QoS_i
global Ci
global Bl
global IR
global RL

global YX

global CCi

connection_num = 6;
link_num = 4;
route_num = 12;
Bl = [ 140 218 161 302 ]';
Ci = [ 7 2 7 3 1 5 ]';
Alpha_i = [ 25.000 25.000 25.000 25.000 25.000 25.000 ]';
Lambda_i = [ 0.250 0.250 0.250 0.250 0.250 0.250 ]';
QoS_i = [ 0.220 0.220 0.220 0.220 0.220 0.220 ]';

IR = [ ...
1 1 0 0 0 0 0 0 0 0 0 0 ;...
0 0 1 1 0 0 0 0 0 0 0 0 ;...
0 0 0 0 1 1 0 0 0 0 0 0 ;...
0 0 0 0 0 0 1 1 0 0 0 0 ;...
0 0 0 0 0 0 0 0 1 1 0 0 ;...
0 0 0 0 0 0 0 0 0 0 1 1 ;...
];
```

```
RL = [ ...
1 0 0 0 ;...
0 1 1 1 ;...
1 1 0 0 ;...
0 0 1 1 ;...
0 1 0 0 ;...
1 0 1 1 ;...
1 0 0 0 ;...
0 1 1 1 ;...
1 1 0 0 ;...
0 0 1 1 ;...
1 0 0 0 ;...
0 1 1 1 ;...
];

% check for solution existence
[check_existance, problematic_cons] = existSolution;
fprintf( 'solution exists = %d\n', check_existance );
if( check_existance == 0 )
    fprintf('No solution found!!!\n');
    return;
end

if( IR == eye(connection_num, route_num) )
   fprintf( '\nIR is an Identity Matrix!\n\n' );
   [fid errmessage] = fopen(output_file, 'w');
   if( fid == -1 )
      fprintf(2, '%s\n\n', errmessage);
   else
      pName = 'net_119_2';
      fprintf(fid, 'Results for problem: %s\n\n', pName);
      fprintf(fid, 'Best function value = 1.0000\n\n' );
      fprintf(fid, 'Number of variables = %d\n\n', route_num);
      fprintf(fid, '%7.4f\n', Ci);
      fclose(fid);

   end;
   return;
```

```
end;

IRnew = [ ...
1 0 0 0 0 0 ;...
0 1 0 0 0 0 ;...
0 0 1 0 0 0 ;...
0 0 0 1 0 0 ;...
0 0 0 0 1 0 ;...
0 0 0 0 0 1 ;...
];

YX = [ ...
 1  0  0  0  0  0 ;...
-1  0  0  0  0  0 ;...
 0  1  0  0  0  0 ;...
 0 -1  0  0  0  0 ;...
 0  0  1  0  0  0 ;...
 0  0 -1  0  0  0 ;...
 0  0  0  1  0  0 ;...
 0  0  0 -1  0  0 ;...
 0  0  0  0  1  0 ;...
 0  0  0  0 -1  0 ;...
 0  0  0  0  0  1 ;...
 0  0  0  0  0 -1 ;...
];

CCi = [ 0 Ci(1) 0 Ci(2) 0 Ci(3) 0 Ci(4) 0 Ci(5) 0 Ci(6) ]';

x_0 = [];
x_L = zeros( (route_num - connection_num), 1);
x_U = [ Ci(1) Ci(2) Ci(3) Ci(4) Ci(5) Ci(6) ]';
A = [ IRnew; RL'* YX ];
b_L = [ zeros(connection_num, 1); -inf * ones(link_num, 1) ];
b_U = [ Ci; (Bl - (RL'* CCi) )];
c_L = [];
c_U = [];
p_f = 'netmodel_f';
p_c = '';
problemName = 'net_119_2';
```

```
% Set the rest of the arguments as empty
setupFile = []; nProblem  = [];
IntVars = [1:(route_num - connection_num)];
VarWeight = []; KNAPSACK = [];
fIP       = []; xIP      = []; fLowBnd = [];
x_min     = []; x_max    = []; f_opt   = []; x_opt = [];

Prob = glcAssign(p_f, x_L, x_U, problemName, ...
                 A, b_L, b_U, p_c, c_L, c_U, x_0, ...
                 setupFile, nProblem, ...
                 IntVars, VarWeight, KNAPSACK, fIP, xIP, ...
                 fLowBnd, x_min, x_max, f_opt, x_opt);

fprintf( 'Computing in the 1st iteration...\n');
Result = glcSolve(Prob);
PrintResult(Result);
f_k  = Result.f_k;
fOld = f_k;

Y = zeros(route_num, 1);
if( f_k > 1000000.000 )    % f_k = Inf
   fprintf( 'EXPLOSION!!!\n' );
   f_k = 1000000.000;
else
   % Number of times with equal f(x) value
   Equal = 0;
   % Set flag for warm start
   Prob.WarmStart = 1;

   % Try at most 10 times extra to get a better solution
   for i = 1:10
       switch (i)
       case 1, fprintf( 'Computing in the %dnd iteration...\n', i+1 )
       case 2, fprintf( 'Computing in the %drd iteration...\n', i+1 )
       otherwise, fprintf( 'Computing in the %dth iteration...\n', i+1 )
       end;

       Result = glcSolve(Prob);
```

```
      PrintResult(Result);
      f_k = Result.f_k;
      if f_k == fOld
         Equal = Equal + 1;
      else
         Equal = 0;
      end
      fOld = f_k;
      if (Equal == 2)
         fprintf( ' DONE!!!\n\n' )
         break;
      end   % switch(i)
    end   % for i

    allpositive = 1;
    m = size(Result.x_k, 2);
    for i = 1:m
      Y = YX * Result.x_k(:,i) + CCi;
      allpositive = 1;

      for j = 1:route_num
        if Y(j) < 0
          allpositive = 0;
          break;
        end  % if
      end  % for j

      if( allpositive == 1 )
        break;
      end  % if
    end  % for i

    if( allpositive == 0 )
      fprintf( 'NO ALL-POSITIVE RESULT WAS FOUND!!\n' );
    end % if

end   % if f_k

[fid errmessage] = fopen(output_file, 'w');
```

```
if( fid == -1 )
  fprintf(2, '%s\n\n', errmessage);
else
  fprintf(fid, 'Results for problem: %s\n\n', Prob.Name);
  fprintf(fid, 'Best function value = %7.4f\n\n', f_k);
  fprintf(fid, 'Number of variables = %d\n\n', route_num);
  fprintf(fid, '%7.4f\n', Y);
  fclose(fid);
end;

% model's objective function in netmodel_f.m
```

B.2 M-Files For the Objective Functions

B.2.1 Objective Function for the Discrete Game Model

The following Matlab file is an implementation of the objective function given by Equation 3.1 in Chapter 3.

```
% model's objective function
function value = netmodel_f(x)

global connection_num
global link_num
global route_num
global Lambda_i
global Alpha_i
global QoS_i
global Ci
global Bl
global IR
global RL
global YX
global CCi

Gfun  = zeros(connection_num, 1);
```

```
Ffun  = zeros(connection_num, 1);
Cir   = zeros(connection_num, route_num);
Cil   = zeros(connection_num, link_num);     % c[i,l]
Cl    = zeros(link_num, 1);
Lones = ones(link_num, 1);
Cr    = zeros(route_num, 1);
Er    = zeros(route_num, 1);
Rho_r = zeros(route_num, 1);
Rones = ones(route_num, 1);
Br    = zeros(route_num, 1);
Bi    = zeros(connection_num, 1);
Iones = ones(connection_num, 1);

%new transformation: x's size is now [route_num-conn_num, 1]
Y  = zeros(route_num, 1);

Y = YX * x + CCi;

index = 1;
for ii = 1:connection_num
   for rr = 1:route_num;
      if( IR(ii,rr) == 1 )
         Cir(ii,rr) = Y(index);
         index = index + 1;
      end  % if IR
   end  % for rr
end  % for ii

value = 0;

% calculate Rho_r
Rho_r = (IR .* Cir)' * (Lambda_i ./ Ci);

% to avoid division by zero, do the following:
for ll = 1:link_num
   if( Bl(ll) <= 0 )
       Bl(ll) = 0.0001;
   end  % if Bl(ll)
end  % for ll
```

```
%compute Cil(i,l), Cr(r), B_r(r) and finally, B_i(i)

Cil = Cir * RL;
Cl = ( sum(Cil, 1) )';

saturated_links = [];
saturated_links = find( Cl >= Bl );

%Cr = min( RL * ( Bl - Cl ) );

for rr = 1:route_num
   Cr(rr) = 10000;
   for ll = 1:link_num
      if( RL(rr,ll) == 1 )
         if( Cl(ll) >= Bl(ll) )
            Cl(ll) = Bl(ll) - 0.0001;
         end % if Cl
         if( Cr(rr) > ( Bl(ll) - Cl(ll) ) )
           Cr(rr) = Bl(ll) - Cl(ll);
         end  % if Cr
      end  % if RL
   end  % for ll
   Br(rr) = Erlang( Rho_r(rr), Cr(rr) );
end  % for rr

Bi = Iones - ( ( Cir * (Rones - Br) ) ./ Ci );
for ii = 1:connection_num
   if( Bi(ii) > QoS_i(ii) )
      Bi(ii) = 0.99 * QoS_i(ii);
   end % if
end  % for ii

% compute the components of the objective function
Gfun = 1 ./ (QoS_i - Bi);
Ffun = Alpha_i .* ( Cil * ( 1 ./ ( Lones - (Cl ./ Bl) ) ) );

tempMat = Gfun + Ffun;
value = sum(tempMat, 1);
```

```
% end   % objectivefun
%-------------------------------------------------------------------
%-------------------------------------------------------------------
function [erlangvalue] = Erlang(rho, cap)
tempprod = 1;
tempsum = 1;
if( cap > 0 )
   for count = 1:cap
        tempprod = tempprod * rho / count;
      tempsum = tempsum + tempprod;
   end
   erlangvalue = tempprod / tempsum;
elseif( cap == 0 )
   erlangvalue = 1;
end  %if
% end Erlang()
%-------------------------------------------------------------------
%-------------------------------------------------------------------
```

B.2.2 Objective Function for the Discrete Revenue Model

The objective function (Equation 4.1)

```
% model's objective function
function value = netmodel_rev_f(x)

global connection_num
global link_num
global route_num
global Lambda_i
global Alpha_i
global QoS_i
global Ci
global Bl
global IR
global RL
global YX
```

```
global CCi

% global X

Gfun = zeros(connection_num, 1);
%Ffun = zeros(connection_num, 1);
Cir = zeros(connection_num, route_num);
Cil = zeros(connection_num, link_num);     % c[i,l]
Cl = zeros(link_num, 1);
Lones = ones(link_num, 1);
Cr = zeros(route_num, 1);
Er = zeros(route_num, 1);
Rho_r = zeros(route_num, 1);
Rones = ones(route_num, 1);
Br = zeros(route_num, 1);
Bi = zeros(connection_num, 1);
Iones = ones(connection_num, 1);

%new transformation: x's size is now [route_num-conn_num, 1]
Y  = zeros(route_num, 1);

Y = YX * x + CCi;

index = 1;
for ii = 1:connection_num
   for rr = 1:route_num;
      if( IR(ii,rr) == 1 )
         Cir(ii,rr) = Y(index);
         index = index + 1;
      end  % if IR
   end  % for rr
end  % for ii

value = 0;

% calculate Rho_r
Rho_r = (IR .* Cir)' * (Lambda_i ./ Ci);

% to avoid division by zero, do the following:
```

```
for ll = 1:link_num
   if( Bl(ll) <= 0 )
       Bl(ll) = 0.0001;
   end  % if Bl(ll)
end  % for ll

%compute Cil(i,l), Cr(r), B_r(r) and finally, B_i(i)

Cil = Cir * RL;
Cl = ( sum(Cil, 1) )';

saturated_links = [];
saturated_links = find( Cl >= Bl );

%Cr = min( RL * ( Bl - Cl ) );

for rr = 1:route_num
   Cr(rr) = 10000;
   for ll = 1:link_num
      if( RL(rr,ll) == 1 )
         if( Cl(ll) >= Bl(ll) )
            Cl(ll) = Bl(ll) - 0.0001;
         end % if Cl
         if( Cr(rr) > ( Bl(ll) - Cl(ll) ) )
           Cr(rr) = Bl(ll) - Cl(ll);
         end  % if Cr
      end  % if RL
   end  % for ll
   Br(rr) = Erlang( Rho_r(rr), Cr(rr) );
end  % for rr

Bi = Iones - ( ( Cir * (Rones - Br) ) ./ Ci );

Gfun = Lambda_i .* Alpha_i .* ( Iones - Bi );

% compute the objective function
value = -1 * sum( Gfun, 1 );

% end   % objectivefun
```

```
%-----------------------------------------------------------------
%-----------------------------------------------------------------
function [erlangvalue] = Erlang(rho, cap)
tempprod = 1;
tempsum = 1;
if( cap > 0 )
   for count = 1:cap
        tempprod = tempprod * rho / count;
      tempsum = tempsum + tempprod;
   end
   erlangvalue = tempprod / tempsum;
elseif( cap == 0 )
   erlangvalue = 1;
end  %if
% end Erlang()
%-----------------------------------------------------------------
%-----------------------------------------------------------------
```

Bibliography

[1] S. Ahn, R. P. Tsang, S. Tong, and D. H. C. Du. Virtual path layout design on ATM networks. In *Proceedings of IEEE INFOCOM'94*, pages 192–200, 1994.

[2] K. Anandalingam, G.: Nam. Conflict and cooperation in designing international telecommunication networks. *Journal of the Operational Research Society*, 48(6):600–611, June 1997.

[3] N.G. Aneroussis and A.A. Lazar. Virtual path control for ATM networks with call level quality of service guarantees. In *Proceedings of IEEE INFOCOM'96*, March 1996.

[4] S. Areibi and A. Vannelli. A grasp clustering technique for circuit partitioning. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science American Mathematical Society*, 35:711–724, 1997.

[5] S. Areibi and A. Vannelli. Tabu search: a meta heuristic for netlist partitioning. *VLSI Design*, 11(3):259–283, 2000.

[6] M. Aydemir and Y. Viniotis. Deterministic algorithm for VP assignment in ATM networks. *Computer Communications*, 19:1036–1050, 1996.

[7] T. Başar and G. Jan Olsder. *Dynamic Noncooperative Game Theory*. Academic press, 1995.

[8] N. Blefari-Melazzi, M. Femminella, and G. Reali. Dynamic bandwidth allocation in a circuit-switched network: provision of deterministic and statistical qos guarantees. In *INFOCOM*, 2000.

[9] J.A. Bondy and U.S.R. Murty. *Graph theory with applications*. Basingstoke Macmillan, London, 1976.

[10] E. Brockmeyer, H.L. Halstrom, and A. Jensen. *The life and Works of A. K. Erlang*. Academy of Technical Sciences, Copenhagen, 1948.

[11] M. Charikar, J. Naor, and B. Schieber. Resource optimization in qos multicast routing of real-time multimedia. In *INFOCOM*, 2000.

[12] I. Chlamtac, A. Faragó, and T. Zhang. How to establish and utilize virtual paths in ATM networks. In *Proceedings of IEEE ICC'93*, pages 1368–1372, 1993.

[13] R. Cocchi, D. Estrin, S. Shenker, and L. Zhang. Pricing in computer networks: motivation, formulation and example. *IEEE/ACM Transactions on Networking*, 1:614–627, February 1993.

[14] C Courcoubetis, A. Dimakis, and M. Reiman. Providing bandwidth guarantees over a best-effort network: call-admission and pricing. In *INFOCOM*, 2001.

[15] N. G. Duffield, P. Goyal, A. Greenberg, P. Mishra, K. K. Ramakrishman, and J. E. Van der Merwe. A flexible model for resource management in virtual private networks. In *ACM, conference on applications, technologies, architecture and protocols for computer communication*, pages 95–108, 1999.

[16] D. Fayek, G. Kesidis, and A. Vannelli. Hierarchical virtual path allocation in larg-scale ATM networks using noncooperative game models. In *Canadian*

Conference on Broadband Research (CCBR). pages 222–233. Ottawa. November 1999.

[17] The ATM Forum. Private network-network interface specification version 1.0. *The ATM Forum*. Mar 1996.

[18] The ATM Forum. Multi-protocol label switching. *The ATM Forum*. 1998.

[19] C. Frei and B. Faltings. Abstraction and constraint satisfaction techniques for planning bandwidth allocation. In *INFOCOM*. 2000.

[20] E. Galperin. The cubic algorithm. *Jounal of Mathematical Analysis and Applications*. 112:635–640. 1985.

[21] R. Garg and H. Saran. Fair bandwidth sharing among virtual networks: a capacity resizing approach. In *INFOCOM 2000*. 2000.

[22] O. Gerstel. I. Cidon. and S. Zaks. The layout of virtual paths in ATM networks. *IEEE/ACM Transactions on Networking*. 4(6):873–884. December 1996.

[23] A. Gibbons. *Algorithmic Graph Theory*. Cambridge University Press. 1985.

[24] R. Guérin and A. Orda. Qos-based routing in networks with inaccurate information: Theory and algorithms. In *IEEE INFOCOM'97*. 1997.

[25] H. Hadama. R. Kawamura. T. Izaki. and T. Tokizawa. Direct virtual path configuration in large-scale ATM networks. In *Proceedings of IEEE INFOCOM'94*. pages 201–207. 1994.

[26] D. M. Himmelblau. *Applied Nonlinear Programming*. McGraw-Hill. New York. 1972.

[27] K. Holmström. *TOMLAB User's Guide*. http://www.ima.mdh.se/tom. 1999.

[28] K. Holmström. Variables redifinition for problem size reduction. In *private correspondance*. 2000.

[29] M.T.T. Hsiao and A.A. Lazar. Optimal decentralized flow control of markovian queueing networks with multiple controllers. *Performance Evaluation*. 13(3):181-204. 1991.

[30] D. R. Jones. C. D. Perttunen. and B. E. Stuckman. Lipschitzian optimization without the lipschitz constant. *Journal of Optimization Theory and Applications*. 79(1):157-181. October 1993.

[31] M. C. Joshi and R. K. Bose. *Some Topics in Nonlinear Functional Analysis*. John Wiley. New York. 1985.

[32] E.P.C. Kao. *An Introduction to Stochastic Processes*. Duxbury Press. Belmont. Calif.. 1997.

[33] K. Kar. M Kodialam. and T. V. Lakshman. Minimum interference routing of bandwidth guaranteed tunnels with MPLS traffic engineering applications. *IEEE Journal on Selected Areas in Communications*. 18(12):2566–2579. December 2000.

[34] F.P. Kelly. *Reversibility and Stochastic Networks*. Wiley. Chichester. 1979.

[35] F.P. Kelly. Blocking probabilities in large circuit-switched networks. *Advances in Applied Probability*. 18:473–505. 1986.

[36] F.P. Kelly. Loss networks. *The annals of Applied Probability*. 1(3):319–378. 1991.

[37] G. Kesidis. *ATM Network Performance*. Kluwer Academic Publisher. 1996.

[38] G. Kesidis. VPC design for ATM networks. In *private discussion*, 1997.

[39] Y.A. Korilis and A.A. Lazar. On the existence of equilibria in noncooperative optimal flow control. *Journal of the Association for Computing Machinery*, 42(3):584-613, May 1995.

[40] Y.A. Korilis, A.A. Lazar, and A. Orda. Architecting noncooperative networks. *IEEE Transactions on Selected Areas in Communications*, 13(7):1241-1251, September 1995.

[41] Y.A. Korilis, A.A. Lazar, and A. Orda. Achieving network optima using stackelberg routing strategies. *IEEE/ACM Transactions on Networking*, 5(1):161-173, February 1997.

[42] Y.A. Korilis, A.A. Lazar, and A. Orda. Virtual path bandwidth allocation in multiuser networks. *IEEE/ACM Transactions on Networking*, 5(6):861-871, December 1997.

[43] Y.A. Korilis and A. Orda. Incentive compatible pricing strategies for QoS routing. In *INFOCOM*, 1995.

[44] Y.A. Korilis, T.A. Varvarigou, and S.R. Ahuja. Incentive-compatible pricing strategies in noncooperative networks. In *INFOCOM*, 1997.

[45] A. A. Lazar, A. Orda, and D. E. Pendarakis. Virtual path bandwidth allocation in multi-user networks. In *Proceedings of IEEE INFOCOM'95*, pages 312–320, 1995.

[46] W. C. Lee. Spanning tree method for link state aggregation in large communication networks. In *IEEE INFOCOM'95*, pages 297–302, 1995.

[47] F. Lin and K-T. Cheng. Virtual path assignment and virtual circuit routing in ATM networks. In *GLOBECOM'93*, pages 436–441. December 1993.

[48] G.M. Louth. *Stochastic Networks: Complexity, Dependence and Routing*. PhD thesis, University of Cambridge, 1990.

[49] D. G. Luenberger. *Introduction to Linear and Nonlinear Programming*. Addison-Wesley, Massachusetts, 1973.

[50] V. Marbukh. Minimum regret approach to network management under uncertainty with application to connection admission control and routing. In *Internation Conference on Networking (ICN)*, pages 309–318, 2001.

[51] V. Marbukh and N. Moayeri. A framework for throughput and stability analysis of a DS-CDMA network. In *IEEE-VTC*, pages 596–600, 1999.

[52] L.G. Mason, Z. Dziong, and N. Tetreault. Fair-efficient call admission control policies for broadband networks. In *IEEE International Conference on Communications '93.*, pages 976–982, May 1993.

[53] R. Mazumdar, L.G. Mason, and C. Douligeris. Fairness in network optimal flow control: optimality of product forms. *IEEE Transactions on Communications*, 1:775–782, May 1991.

[54] E. J. Messerli. Proof of a convexity property of the Erlang B formula. *Bell Systems Technology Journal*, 51(4):951–953, 1972.

[55] A. Migdalas, P. M. Pardalos, and P. Värbrand. *Multilevel Optimization: Algorithms and Applications*. Kluwer Academic Publishers, Dordrecht, The Netherlands, 1998.

[56] R. Mladineo. An algorithm for finding the global maximum of multimodal, multivariate function. *Mathematical Programming*, 34:188–200, 1986.

[57] M. Montgomery and G. De Veciana. Hierarchical source routing through clouds. In *IEEE INFOCOM'98*, 1998.

[58] S. Ohta and K. I. Sato. Dynamic bandwidth control of the virtual path in an asynchronous transfer mode network. *IEEE Transactions on Communications*, 40(7):1239–1247, July 1992.

[59] A. Orda, R. Rom, and N. Shimkin. Competitive routing in multi-user communication networks. In *INFOCOM'93*, volume 3, pages 964–971, 1993.

[60] M.J. Osborne and A. Rubinstein. *A Course in Game Theory*. MIT Press, Cambridge, Massachusetts and London, England, 1994.

[61] J. Pinter. Globally convergent methods for n-dimensional multiextremal optimization. *Optimization*, 17:187–202, 1986.

[62] A. Pitsillides, G Stylianou, C.S. Pattichis, A. Sekercioglu, and A. Vasilakos. Bandwidth allocation for virtual paths (BAVP): investigation of performance of classical constrained and genetic algorithms based optimisation techniques. In *INFOCOM 2000*, 2000.

[63] K.W. Ross. *Multiservice Loss Models for Broadband Telecommunication Networks*. Springer-Verlag, 1995.

[64] S. Sarkar and L. Tassiulas. Fair allocation of discrete bandwidth layers in multicast networks. In *INFOCOM*, 2000.

[65] B. Shubert. A sequential method seeking the global maximum of a function. *SIAM Journal on Numerical Analysis*, 9:379–388, 1972.

[66] R. Siebenhaar. Optimized ATM virtual path bandwidth management under fairness constraints. In *Proceedings of IEEE GLOBECOM'94*. pages 924-928. 1994.

[67] R. Siebenhaar. Multiservice call blocking approximation for virtual path based ATM networks with cbr and vbr traffic. In *Proceedings of IEEE INFOCOM'95*. pages 321-329. 1995.

[68] D.R. Smart. *Fixed Point Theorems*. Cambridge University Press. London. 1974.

[69] N. Spring. M. Chesire. V. Sahasranaman. and B. Anderson. T. Bershad. Receiver based management of low bandwidth access links. In *INFOCOM*. 2000.

[70] E. Sullivan and R. Callon. P-nni draft specification. *ATM Forum*. July 1994.

[71] R.W. Wolff. *Stochastic Modeling and the Theory of Queues*. Prentice Hall. Englewood Cliffs. N.J.. 1989.

[72] C. Xinjie and K. R. Subramanian. An optimal admission control scheme for the wireless ATM networks. In *NOM2000*. 2000.

[73] H. Yaïche. R. Mazumdar. and C. Rosenberg. Distributed algorithms for fair bandwidth allocation to elastic services in broadband networks. In *INFOCOM*. 2000.