INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

ProQuest Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600

Université d'Ottawa • University of Ottawa

Design of Cost-Efficient Multi-Service Internet Backbones

By

Jia, Qihui

A thesis submitted to
The School of Graduate Studies and Research
University of Ottawa
In partial fulfillment of the requirements
For the degree of

M. Sc of Systems Science

Master's in Systems Science Program
University of Ottawa
Ottawa, Ontario, CANADA

December 2002

Abstract

One of the main objectives when designing a Multi-Service Internet is to define a profitable and efficient network configuration. Network profit is a very important optimization objective for today's Internet Service Providers (ISP). However, successful optimization for a large system such as a nation wide Internet backbone supporting both voice and data services requires a fast-responding and high-efficient method. In this thesis, we explore the methodology for such optimization design and present a two-phase method. In the first phase, a mathematical method based on a Linear Programming (LP) solution technique is utilized as primitive method with simplified objective functions and constraints. A Hopfield Neural Network (HNN) model is applied in the second phase to get an extended optimization for the network performance aiming to effectively distribute the network traffic across the network. The method has been developed and applied to the design of a national Internet backbone network. Both models are formally defined, optimally formulated, and solved.

Keywords

Optimization, multi-service, delay, capacity, linear programming, Hopfield neural network

Acknowledgements

I would like to thank my supervisor, Professor Luis Orozco-Barbosa, for his advice and comments on my work, and his suggestions that helped to improve this report. All errors are of course my own.

A special gratitude also goes to my friend Dianne Dinsdale who always helped and encouraged me with positive affirmation.

Finally, many thanks go to my parents and my husband, Yuan, for their unfailing support and encouragement.

Contents

List of Figures

List of Tables

List of Acronyms

Initials	Meanings
BVPN	Broadcast Virtual Private Network
CHNN	Continuous Hopfield Neural Network
CPLEX	A linear programming solver tool
GA	Genetic Algorithm
GUI	Graphic User Interface
HNN	Hopfield Neural Network
IP	Internet Protocol
ISP	Internet Service Provider
Kbps	kilobit per second, traffic rate unit
LAN	Local Area Network
LP	Linear Programming
MAN	Metropolitan Area Network
MCF	Multi-commodity Flow
MPLS	Multi-Protocol Label Switching
NDP	Network Dimensioning Problem
NN	Neural Network
OSPF	Open Shortest Path First
PSTN	Public Switched Telephone Network
QoS	Quality of Service
WAN	World Area Network
VPN	Virtual Private Network

List of Symbols

Symbols	Meanings
N	the set of nodes
L	the set of links
L'	the set of undirected links
(s,d)	a pair of nodes (source, destination)
A	the set of arcs between two nodes (s,d)
R	the set of admission route paths between node pair (s,d)
$G(N,L)$	the transmission layer supply graph
$G(N,L')$	the associated graph
$G(N,A)$	the multigraph layer graph
$G(N,R)$	the routing layer graph
$R(s,d)$	the set of preliminary routes between (s,d). In some formulations, it also represents the number of the elements in the set
k	the constant cost of a capacity unit which is positive
n_{ij}	the bound of the number of capacity unit on the link (i,j) for voice service
n_{ij}^{v}	the number of capacity units loaded on the link (i,j) for voice service
δ_{sd}^{v}	the earning rate of voice service per capacity unit over (s,d)
$F_{(sd)\gamma}^{v}$	the total flow of voice packets on service route where γ is a route (bps)
δ_{sd}^{d}	the earning rate of data service over (s,d)
F_{ij}^{d}	the total flow of data service on link (i,j) (bps)
ρ_{sd}^{d}	the end-to-end blocking probabilities for data service
n_{ij}^{d}	the number of capacity unit loaded on the link (i,j) for data service

τ_{ij}^{d}	equals 1 if i is the source city of node pair (s,d)
$T(T_{sd})$	a $N \times N$ traffic matrix of bandwidth demands from source node to destination node (bps)
α	the service multiplier for voice service
F_{ij}	the total packet flow over each link from i to j (bps)
$\beta_{(sd)\gamma(ij)}$	an indicator, if the γth route between (s,d) includes the $ijth$ link, then equals 1, else equals 0
C_{ij}	the capacity of each link from i to j (bps)
C_u	the capacity unit (bps)
del_{ij}	the maximum acceptable delay of link (i,j) (ms)
P	the average packet size transmitted over a link (bit)
ρ	the traffic density
μ	the packet arrival rate
μ'	the service rate
c_{ij}	the capacity of link (i,j) (in message/sec)
f_{ij}	the average traffic rate on link (i,j)
F_{sd}	the total data traffic flow between (s,d) (kbps) which is obtained from the first phase optimization
$\chi_{sd\gamma}$	the proportion of the traffic between (s,d) transferred through the γth route
l	the lth link
$\beta_{(sd)\gamma l}$	an indicator, if the γth route between (s,d) includes the lth link, then equals 1, else equals 0
ad_l	the maximum acceptable average delay of the lth link (s)
c_l	the capacity of the lth link (kbps), also obtained from the first phase optimization
$f(x)$	$f(x)=\begin{cases} x, & \text{if } x \geq 0 \\ 0, & \text{if } x < 0 \end{cases}$

Chapter 1

Introduction

1.1 Motivation

The great success of the Internet has spurred an exponential growth on the network resources required to fulfill the traffic demand. Future Internet communication systems will have to provide a wide variety of sophisticated services, e.g. Internet telephony, video on demand, premium data, as well as best-effort service [3][20]. Under this scenario, high-bandwidth and delay-sensitive payloads like voice and video have to be carried over the same network infrastructure sharing resources with other time-insensitive services, i.e., data [13]. Contrary to the higher tolerance to delay of data services that can be either connection oriented or connectionless, most voice services are inherently connection-oriented and aimed to provide service guarantees to the end user [4]. This latter feature represents a big challenge towards the integration of voice and data services.

In addition, the rapid increase on the number of users often causes congestion, which can result in delays and higher loss rates in the packet transmission. Moreover, because the Internet is a packet-switched or connectionless network, each signal is digitized into individual packets and sent over separate paths before reassembled at the ultimate destination [6]. Although such mechanism makes more efficient use of the network resources than circuit-switched network (PSTN), it also increases the potential of packet delay and loss [27].

Due to the aforementioned issues, nowadays the Internet Service Providers (ISPs) have to implement and manage Quality-of-Service (QoS) controls in a cost-efficient manner to meet the user's demands. An important objective on the design of the future Internet system is to achieve the lowest capacity cost and offer higher link capacities while guarantying various levels of quality of service.

Nowadays, the efficient and optimal design of a network, which is necessary condition to make a communication networks usable and affordable, has been an active area of research, and been studied using a wide variety of optimization models and objectives. The primary issue remaining in most previous studies is that researches have not provided a comprehensive analysis on the design of future Internetworking systems. In particular, the following issues still have to be addressed:

1. Most studies have only focused on networks offering loss-less data services. However, multiple heterogeneous services will be provided by future Internetworking systems, each service being characterized by its own QoS requirements.

2. Not all the design properties of objective functions can be achieved through traditional mathematical solvers. A much faster response of control algorithms of future Internet must be guaranteed according to the extremely high transmission rate. This requires the use of methodologies in addition to LP with fast convergence capabilities.

3. The integration of the network performance and fast routing control requirements have not been addressed in the optimal design of multi-service national backbone networks, whereas such integration is a very important issue in such network infrastructures.

Hence further investigations in optimal models and solutions of a nation-wide Internet with abilities of providing various services at the same time are complementary to previous research efforts. These models and solutions may be multifarious. Moreover, an approach that optimizes the overall network profit with the combination of Linear Programming (LP) and Neural Network (NN) models is a worthy task. So far our description of the objectives is similar to the relative network dimensioning problems. But we successfully include all the issues above into our concern.

1.2 Objective and Contributions

This thesis presents a two-phase design for a multi-service nation-wide Internet backbone. In particular we focus on the development of the Linear Programming (LP) model and the Hopfield Neural Network (HNN) model to optimize the network profits, as well as the network performance. We demonstrate the idea in the nation-wide Internet backbone of Colombia. Also we formally implement both of these two models. The Linear Programming (LP) solver CPLEX is utilized in the first phase. And the end, a friendly Graphic User Interface (GUI) has been designed to facilitate the users in the second phase.

The results and contributions of the thesis are:

1. A model-based approach for a cost-efficient nation-wide Internet backbone system design. The method is a new idea to successfully combine Linear Programming (LP) and Hopfield Neural Network (HNN) in a network dimensioning problem, which takes the advantages of both techniques.

2. The requirements and the solutions are presented in full mathematical models to provide a step by step procedure for it. And with a self-developed GUI, different experiments and applications of sensitive analysis are easily handled by users.

3. The thesis also includes a routing path picking-up algorithm which is based on the algorithm in Ref [15] in order to reduce the load of the traffic traversing over each path.

1.3 Thesis Outline

The thesis is organized as follows. Chapter 2 provides an overview of Linear Programming (LP) and Hopfield Neural Network (HNN) techniques which are used throughout this thesis. It also includes as introduction to the issues when designing a nation-wide Internet backbone. Chapter 3 describes the proposed methodology to solve the optimization problem. Chapter 4 applies the proposed methodology to the design of a nation-wide Internet backbone system. Chapter 5 draws our conclusions and overviews the topics for further investigation.

Chapter 2

Background

This chapter should provide the readers with the basic concepts used throughout the thesis. We proceed with an overview of the network dimensioning problem and Internet backbone system, and then introduce the concepts of the methods used, namely Linear Programming (LP) and Hopfield Neural Network (HNN).

2.1 Previous Research in Network Optimization

Network optimization has been the subject of interest for a long time. A large number of works have been reported in the literature [3,5,7,8,10,15,17,20,26]. Multi-commodity flow (MCF) models have been widely used to formulate the problems in different networks, such as VPN [5][7], LAN/MAN [8][10][26], MPLS network [3], wireless network [20], and packet-switched network [15][17].

There are several approaches which can be used to build the optimization models with different objectives. They include Linear Programming (LP), Neural Network (NN), Genetic Algorithm (GA), Stochastic Process model, and some self-proposed heuristic algorithms. Among them, Linear Programming (LP) has been adopted by most studies. In [1], M. Plante and B. Sansò provide an excellent review on the work done in this area. Most proposals are largely aimed at minimizing the capacity cost since it is the most common consideration when building a network. In [7], the assigned capacity cost over a

BVPN has been minimized through the combination of the LP model and a heuristic algorithm.

In [8] and [10], the authors also look into the optimization of the cost of the installed capacity for MANs. While in [3][5], the authors propose a throughput-based optimization in order to maximize the total network revenue for multi-service networks. In their research, they develop a stochastic process model to supplement a Linear Programming optimization and make well consideration about the multi-service integration. However, they did not take the QoS factors into account. In addition, although in order to guarantee the service of the time-sensitive applications which limited the other service performance, there is no further optimization or consideration for better performance of them.

Some other researches take more consideration on the performance requirements and target at minimizing the network delay [15][26] or congestion [17]. Differently, [26] uses LP and GA to solve the delay minimization problem in order to get the optimal topology for a LAN, while the other two use NN techniques. In [17], J.E. Wieselthier and A. Ephremides proposed a new way in defining the neuron of Hopfield neural network which was also used by G. Feng et al [32]. The former aimed at finding the lowest congestion for radio networks and the latter tried to reduce the total delay of a packet-switched network. But both of their models only include the constraints generated by the neuron characteristics. The Hopfield neural network has also been adopted in [20] to perform the optimization for the installation cost of a mobile access network.

2.2 Techniques for Network Optimization

2.2.1 Linear Programming (LP)

Linear programming is an important and very useful optimization technique because many practical problems can be formulated as LP's and the existing algorithms enable us to solve these problems numerically relatively easily. A linear program is the

minimization or maximization of a linear function subject to some linear constraints [19]. More explicitly, such problem with n variables and m constraints typically has the form:

$$\begin{aligned} &\textit{maximize} && \left\{\sum_i c_i x_i\right\} \\ &\textit{subject to} && \sum_i A_{1i} x_i \le b_1 \\ & && \sum_i A_{2i} x_i \le b_2 \\ & && \alpha_i \le x_i \le \beta_i \qquad \textit{for each } i \in \{1...n\} \end{aligned}$$

If all the variables take on real values, the problem can be solved using a linear programming technique. This ensures that the problem can be solved in polynomial time. In practice linear programs can be solved efficiently for reasonable-sized problems, or even for large problems having a special structure. However when some or all of the variables take only on integer values, corresponding to a pure integer or a mixed integer programming respectively, the problem becomes *NP*-complete.

Linear Programming (LP) and its extensions are valuable for modeling various types of problems in planning, routing, scheduling, assignment, and design. The use of LPs in transportation, energy, telecommunications, and manufacturing are widely-used in the industry. There is a large selection of software tools for modeling and solving LP problems. Among them, CPLEX is a high-performance, robust, flexible optimizer for solving linear, mixed-integer and quadratic programming problems in mission-critical resource allocation applications. For these reasons, we have used CPLEX in our research efforts.

2.2.2 Hopfield Neural Network (HNN)

A feedback network is also used to implement optimization problems. These networks interconnect the neurons with a feedback path. The Hopfield Neural Network (HNN),

first demonstrated by Hopfield and Tank [33], is a typical instrumental one based on Lyapunov energy function. Because of its simple architecture and powerful computational ability, it has been studied extensively during the past decades. The following description is based on material from [35][15][16] and [18].

The Hopfield networks are recurrent neural networks, in which the output values are fed back to the input of the network in an undirected way. The network structure is built in a way that the outputs of each neuron are connected to the input of every other neuron, as depicted in Figure 2.1 [31]. The Hopfield net is a fully connected network and no distinction is made between input and output units. It consists of a set of individual processing units called neurons with summations and non-linear activation functions.

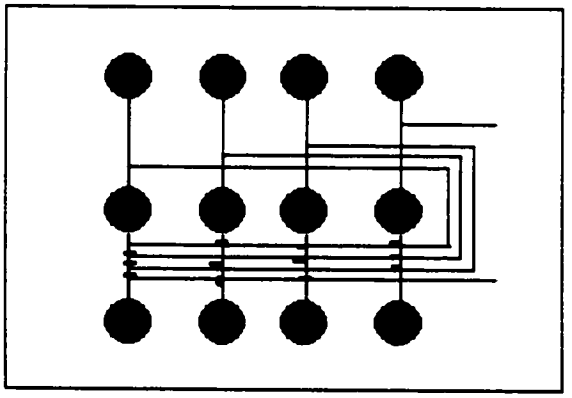

Figure 2. 1 ***The Hopfield Network*** [31]

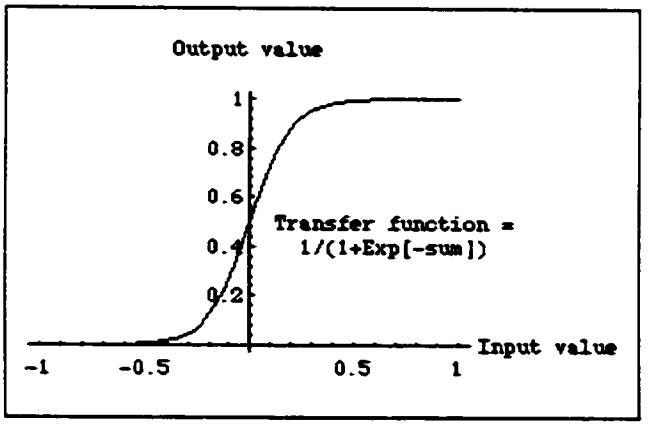

Figure 2. 2 ***Sigmoid Transfer Function*** [34]

In general each neuron is described by two characteristics: activation and output. As shown in Figure 2.2, the output of the neuron has a limitation between [0, 1], and the activation function can be the sigmoid function, the piecewise function, or the linear threshold function. If a given problem can be expressed as:

$$E = -(\frac{1}{2}x^T w x + b^T x)$$

where

- X input vector(packet assignments)for which an optimal solution is sought
- b bias vector determined by constraints
- W symmetric weight matrix for the neural network

By employing the Lyapunov theorem, the energy function can be defined as

$$E = -\frac{1}{2}\sum_i \sum_{j \neq i} v_i w_{ij} v_j - \sum_i I_i v_i$$

where v_i, the neuron input-output relation, is typically depicted in the sigmoid function

$$v_i = g(u_i) = \frac{1}{2}[1 + \tanh(\frac{u_i}{u_0})]$$

Hopfield proposed two types neural network, one of which is binary neuron network and another one is continuous neuron network. The Continuous HNN (CHNN) allows the embedding of discrete problems in a continuous solution space, which yields better solution than the digital one. So the CHNN net is often used to be an analog approximation. The dynamics of it usually has the following form in differential equation:

$$\frac{du_i}{dt} = -\frac{u_i}{\tau_i} + \sum_j T_{ij} v_i + I_i$$

Hopfield has also provided in [19] the guidelines on how to choose the values of T_{ij} so that the equation above represents the dynamics corresponding to a given energy function. If the energy function corresponds to an optimization objective, the

initialization of the u_i's to an initial configuration will result in an equilibrium state which settles to a local minimum of the objective function.

2.3 Nation-Wide Backbone Internet

Internet Backbone is the physical network which carries Internet traffic between different networks. Such networks usually rely on fiber optical cable. Because data travel at the speed of light, any place connected to any of the backbone networks should be as accessible as any other place. As a result, Internet Backbone is usually having a high capacity which is measured in megabits per second. The Colombian Internet Backbone is shown in Figure 2.3.

Figure 2. 3 ***Columbia Internet Backbone*** [37]

Chapter 3

Two-phase Cost-efficient Design in Internet Backbones

As introduced in Chapter 2, the network optimization problems include several basic objectives, i.e., cost-based, network revenue-based and throughput-based optimizations. In this chapter, we give a description of our problem which aims at finding the optimum profit for a nation-wide Internet system supporting both voice and data services, subject to constraints on capacity, end-to-end packet delay among others. We then suggest a two-phase method design. The goal in the first phase is to get a maximum profit and an efficient status for the voice service, while in the second phase we try to track the routes from the corresponding sets of predetermined routes for the data packets so that the performance of the whole Internet backbone can be optimal. The related issues such as the identification of objective function and constraints of LP model, construction of HNN energy function, and solution structure are discussed in detail.

3.1 Methodology

Figure 3.1 provides a graphical overview of our methodology. The current layout of the Internet backbone system is the known factor. To achieve the goals, profitable Internet and best performance for each service class, LP technology is used for fundamental calculation in the first phase. With the outputs obtained from the early stage being the inputs of the second phase, an HNN approach is exploited for further network optimization so that the data service can also be provided in an efficient manner.

3.1.1 Three Layered Network System Definition

Networks can be viewed as systems consisting of three layers in which a layer is embedded into another [12][9]. We represent these three layers by three interrelated graphs which are shown in Figure 3.2:

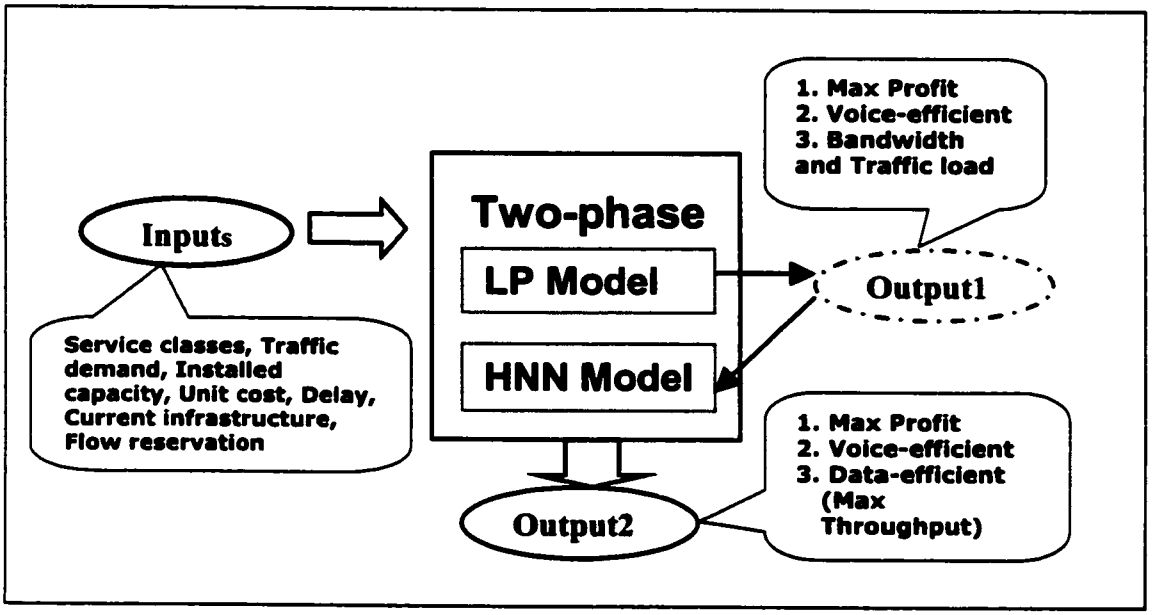

Figure 3. 1 ***Methodology Overview***

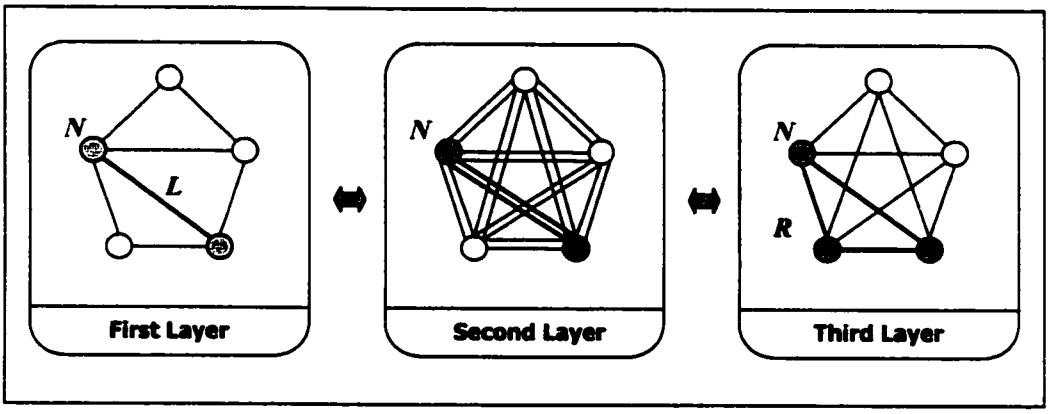

Figure 3. 2 ***Three-Layered System Overview***

1) The first layer is the transmission layer that is represented by a supply graph $G(N, L)$, where N is the set of nodes and L is the set of links. This layer describes the facilities of the network. There is an associated graph $G(N, L')$ with it, where L' specifies the set of undirected links.
2) The second layer is a multigraph with simple paths denoted by $G(N, A)$, where A is the set of arcs between two nodes (s,d).
3) The third layer--routing layer is an equivalent multigraph represented by $G(N, R)$, where R is the set of admission route paths between node pair (s,d).

3.1.2 High Level Problem Definition

The problem to be addressed in the thesis is an Internet system-dimensioning problem. To be able to accomplish it, we should first define the objective function in terms of the cost and revenue, and specify the constraints that derived from the requirements of the performance and survivability. In the early stage of the research, a large number of points had to be considered. The following list provides the main design issues that had to be addressed before setting up the models.

1) How many service classes of data should be classified and included into our approach? What are the differences between these service classes?
2) At which level the variability is taken into account in our approach?
3) Which kind of network dimension strategies is most suitable for our problem?
4) What performance and survivability points are major requirements that must be concerned in the optimization models?
5) How to choose the packet-queuing model for the network in order to make it easier to be described with math formulations?
6) Which NN model is suitable for the second phase? What advantages the second phase model can take from the first LP model in optimization? What are the inputs for the two-phase model?
7) What algorithms are needed for the models and the data preparations?

From the list above it can be seen that many possible model structures can be constructed. In the following text, we provide the details on building the models.

First of all, in current Internets, multiple services are integrated. Different classes of services, such as Internet telephony, video or premium data service, are supported in a converged data network. They have to share the limited bandwidth while using different protocols because of their own characteristics and requirements. Our research only considers two types of services: voice service class and data transmission application. However, it should be clear that our approach is not limited to only these two particular services. We have selected these two service classes because a) they are the two most popular services; b) they belong to two different service classes and have quite different requirements—voice transmission is time-sensitive and mission critical while data transmission has higher throughput requirement but no tight time constraints.

In the first phase, similar formulations are proposed for simulating both classes of services allowing us to simplify the mathematical model expressed in the linear programming formulation. Based on the traffic control strategies and switching strategies, in order to minimize the node complexity and costs, a centralized end-to-end strategy is picked up from five model types to handle the bandwidth allocations and requirement constraints [1]. All the links between the identical source and destination pair (s,d) will be substituted by the end-to-end path, which is a direct logical link. In this way, there are no packets routed through intermediate connection switches because the path is predetermined. Hence, we use R and (s,d) to represent such end-to-end constraints. This important issue is relaxed in the second phase. Since there is no common simulation model for the network optimization with NN model, we introduce a simple way to take the flow-based variability into account by considering the influence from the overall traffic flow distribution.

The performance requirements derive from the preference of the two different service classes under study. Service preferences yield the traffic load that will be associated with

admission routing technique chosen, bandwidth assigned, and acceptable end-to-end delay bounded. In essence, voice services require a relatively modest transmission rate but stringent real-time requirements. Because of its time-sensitive, the limitation of the hops in transmission routes is another requirement to consider. Therefore, we handle the total flow of voice in service-routes accomplished with a preliminary routing table. While in the case of data service, it can tolerate more delay. We try to obtain a possibly highest throughput. The total flow between each source-destination pair is decomposed by reducing all the links with zero flow components.

The survivability requirements dealing with the network failure are becoming more and more important in the efficient high-speed backbone network. When one link happens to fail, there should be at least one alternative link to substitute it for routing the packets. Employing the reservation strategy [9] with Open Shortest Path First (OSPF) routing protocol, we specify a block probability parameter for the traffic demand when we set the flow constraints.

Taking into account the aspects outlined above, a general problem statement may be the following. The total profit to be made is defined as $p = (total\ revenue - total\ costs)$. The cost function is estimated by a linear relationship with the number of the capacity units allocated to the multiple arcs between two nodes, subject to the transmission capacity constraints and bandwidth requirements. The revenue function for the network may consist of the following parts: the revenue for the voice service, and the revenue for the data service. The constraints of this problem are the same as in the cost objective, with one addition, the constraint of the performance requirement. Here we formulate it with the relationship of the average link delay and total flow to link capacity.

The problem to be considered in the second phase may be outlined in the following statement. The final network configuration and the assigned bandwidths for the data service are obtained from the LP optimization. With these results being the inputs so that the obtained optimal status are not changed, the further calculation for the proportion of

the data traffic over each route is taken with a Hopfield Neural Network (HNN) model in order to get the highest throughput and reduce the traffic load over each path. We will compare the best and the worst case to evaluate the model.

3.2 First Phase: LP Based Profit Optimization

The focal point in our work and in a sense the difference from the pertinent researches (see Chapter 2) is that a more general problem version is considered, since it spans over both the voice service and the data service by considering their delay and loss requirements, and relative priorities to guarantee the real-time constraints of the voice service. This section provides the formal statement of the version of the profit optimization problem addressed in this thesis. As aforementioned, LP is a conceptually typical and practical mathematical methodology suitable to our problem. In the following we present our LP model.

3.2.1 Solution Structure

The model structure is depicted in Figure 3.3. The LP model includes an objective function and several constraints. The layout of the whole Internet backbone, the traffic demand, installed bandwidth, routing protocol, as well as the QoS constraints of our model are the input vectors, while the actually enabled capacities and traffic assigned over each directed link or logic link are the output vectors. The performance requirements control the number of iterations when looking for the optimal solution.

3.2.2 Objective Function

In previous works [3][8][10][12][15][21], different objective functions have been used to describe the network performance. However, from the economic benefit aspect, the optimization of the total profit could be the most representative goal for a nation-wide Internet.

We use a Poisson arrival process to model the network traffic and assume an M/M/1 queuing model for the packets of both service classes, since this is a most typical theoretical simplicity for smoothing the total traffic [4] and very analyzable.

Furthermore, since our objective is to optimize a nation-wide Internet, we divide the large backbone system into several clusters, each of which consists of one or two large cities, several medium cities and small cities. As shown in Figure 3.4, these medium and small cities are interconnected to the network via a large city. Due to the few users and little upstream traffic requirements in small cities, they are treated as subscribers of larger cities. Therefore, we do not have to deal with admission and routing issues at these nodes.

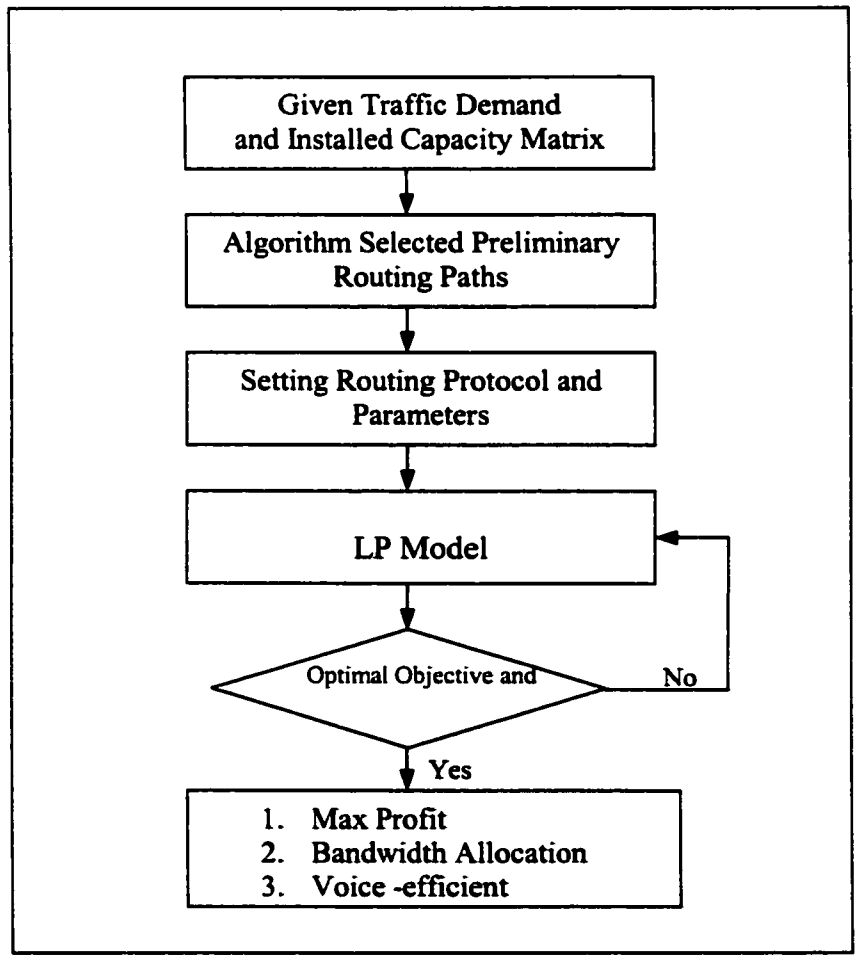

Figure 3. 3 *First-phase Solution Structure*

The above considerations make it easier for us to build the objective functions. In particular, the derivation of maximum profit is calculated in this phase. We give the formulation in two parts: first for the voice service in isolation in order to meet the QoS requirements better, second by combining both service classes. By defining the following notations:

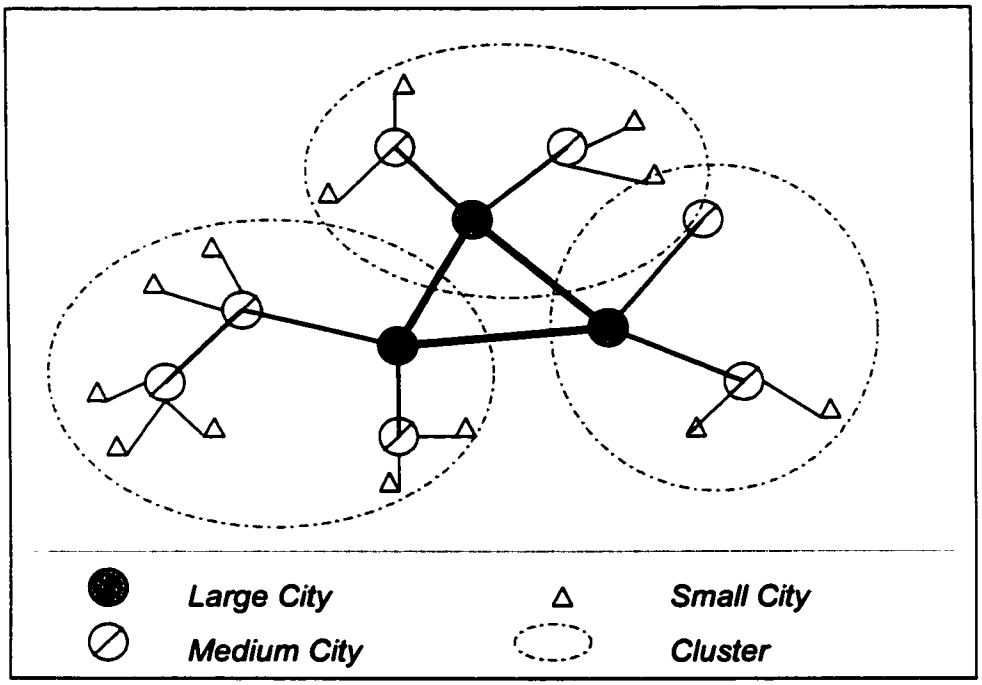

Figure 3. 4 *Example of Cluster Structure*

(s,d) : a pair of nodes (source, destination).

$R(s,d)$: the set of preliminary routes between (s,d). In some formulations, it also represents the number of the elements in the set.

k : the constant cost of a capacity unit which is positive.

n_{ij}: the bound of the number of capacity unit on the link (i,j) need voice service.

n_{ij}^{v}: the number of capacity unit loaded on the link (i,j) for voice service.

δ_{sd}^{v}: the earning rate of voice service per capacity unit over (s,d).

$F_{(sd)\gamma}^{v}$: total flow of voice packets on service route where γ is a route (bps).

δ_{sd}^{d}: the earning rate of data service over (s,d).

F_{ij}^{d}: total flow of data service on link (i,j) (bps).

ρ_{sd}^{d}: end-to-end blocking probabilities for data service.

n_{ij}^{d}: the number of capacity unit loaded on the link (i,j) for data service.

τ_{ij}^{d}: $\tau_{ij}^{d} = \begin{cases} 1 & i \text{ is the source of a node pair } (s,d) \\ 0 & \text{otherwise} \end{cases}$.

the first objective function can thus be described as:

$$\text{maximize} \quad \{ \sum_{(s,d)\gamma \in R(s,d)} \sum \delta_{sd}^{v} F_{(sd)\gamma}^{v} \} \tag{3.1}$$

and the following is the combination objective function:

$$\max \left\{ \max \sum_{(s,d)} \sum_{\gamma \in R(s,d)} \delta_{sd}^{v} F_{(sd)\gamma}^{v} + \sum_{(s,d)(i,j) \in L(s,d)} \sum \delta_{sd}^{d} F_{ij}^{d} \tau_{ij}^{d} (1 - \rho_{sd}^{d}) - \sum_{(i,j) \in L} k \, (n_{ij}^{v} + n_{ij}^{d}) \right\} \tag{3.2}$$

When evaluating the actual capacities assigned to each link, we divide the capacity into several fixed-size capacity units. Therefore in the last term of equation (3.2), the total cost of the network can be estimated by the summation of the bandwidth used on each link multiplied with the constant k . Equation 3.1 represents the total network revenue for the voice service, which is obtained by adding up the products of all the earnings. The second term in Equation 3.2 represents the total data service revenue, which is expressed as the summation of the earning produce by the data flow over each node pair (s,d) with concerning the block probability. Specially and simply we assume the earnings per unit carried bandwidth for each class of service are same for the same (s,d) and the revenue obtained for each admission control service.

3.2.3 Mathematical Formulations for Constraints

Now we express the set of constraints. The following notation is used through the formulation:

$T(T_{sd})$: a $N \times N$ traffic matrix of bandwidth demands from source node to destination node (bps).

α: service multiplier for voice service.

F_{ij}: the total packet flow over each link from i to j (bps).

$\beta_{(sd)\gamma(ij)}$: an indicator, if the γth route between (s,d) includes the $ijth$ link, then equals 1, else equals 0.

C_{ij}: the capacity of each link from i to j (bps).

C_u: the capacity unit (bps).

del_{ij}: maximum acceptable delay of link (i,j) (ms).

P: the average packet size transmitted over a link (bit).

3.2.3.1 Service multiplier

For convenience, we use a single base traffic demand matrix $T(T_{sd})$ for both service classes and apply a service multiplier α for voice service. Obviously, traffic demand matrix for voice service is αT, and $(1-\alpha)T$ for data service. Therefore, we can generate the following constraints:

$$F^{v}_{(sd)\gamma} \geq 0 \tag{3.3}$$

$$F^{d}_{ij} \geq 0 \tag{3.4}$$

$$\sum_{\gamma \in R(s,d)} F^{v}_{(sd)\gamma} \leq \alpha T_{sd} \tag{3.5}$$

$$\sum_{i \in N \mid (i,j) \in L(s,d)} F_{ij}^{d} - \sum_{i \in N \mid (j,i) \in L(s,d)} F_{ji}^{d} = \begin{cases} (1-\alpha) T_{sd} & i = sour \\ -(1-\alpha) T_{sd} & i = dest \\ 0 & otherwise \end{cases} \quad (3.6)$$

The left side in Equation 3.6 gives the total flow of data packets between a pair of node (s,d). When no node on a link that is crossed by a (s,d) pair is either the source or the destination, the subtraction is equal to 0 under the assumption that no traffic is dropped or added at intermediate nodes out or into the network.

3.2.3.2 Survivability requirement

In our case, the predetermined routes with no more than 3 intermediate nodes (we will discuss it in more detail of it later) are listed in Table 4.2. In this table, all the routes between the identical source and the identical destination (s,d) are trying to cross different links which make the whole Internet still be in a safe state in case some links failed.

3.2.3.3 Performance requirement

The voice service is given a higher priority in transmission in order to guarantee almost no loss and real time quality. We just use a slack in Equation 3.4 to represent the possible loss, and apply all the capacity to do the optimization separately for it. In the other hand, an end-to-end blocking probabilities ρ_{sd}^{d} is introduced to the limited possibility of data packet loss.

Furthermore, we include the consideration of the average delay for each link. Now we present the construction of the capacity constraint. An assumption of M/M/1 queuing model for packet arrival of each implemented link has been mentioned in last subsection. The conceptual structure of this typical mathematical model is shown in Figure 3.5. The relationship of the traffic density ρ, the packet arrival rate μ, and the service rate μ' is $\rho = \frac{1}{\mu - \mu'}$. Thus we can yield the following expression for the average delay

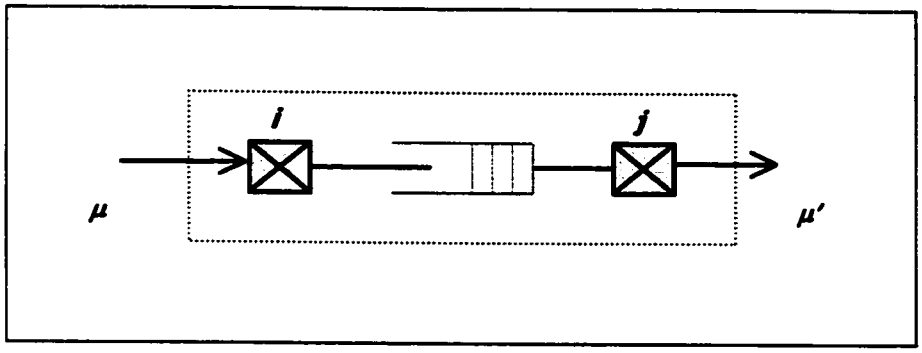

Figure 3. 5 *M/M/1 Queuing Model*

$$delay_{ij} = \frac{1}{c_{ij} - f_{ij}} \tag{3.7}$$

where C_{ij} is the capacity of link (i, j) (in message/sec), and f_{ij} is the average traffic rate on link (i, j). The next step is to find an appropriate expression for the performance constraint. We know the messages and packets in the network maintain almost in a fixed length when they are transferred from a node to another. So we modify the equation into an approximate inequality:

$$del_{ij} \geq 1 \Big/ \frac{C_{ij} - F_{ij}}{P} \tag{3.8}$$

Here F_{ij} consist two components: the flow of voice $F^{v}_{(sd)\gamma}$ and the flow of data F^{d}_{ij}. From the inequality 3.8, we can easily yield the capacity constraints as follows:

$$\sum_{(s,d)\gamma \in R(s,d)} \sum F^{v}_{(sd)\gamma} \beta_{(sd)\gamma(ij)} + \frac{P}{del_{ij}} \leq C_u n_{ij} \tag{3.9}$$

$$\sum_{(s,d)\gamma \in R(s,d)} \sum F^{v}_{(sd)\gamma} \beta_{(sd)\gamma(ij)} + \sum_{(s,d):(i,j) \in L(s,d)} F^{d}_{ij} + \frac{P}{del_{ij}} \leq C_u (n^{v}_{ij} + n^{d}_{ij}) \tag{3.10}$$

where

$$C_u (n^{v}_{ij} + n^{d}_{ij}) \leq C_{ij}$$

$$C_u n_{ij} \leq C_{ij}$$

Additionally, according to the characteristics of voice over Internet, the upstream and downstream carried bandwidths should be in equality because when an IP call is made, a virtual channel with same amount of bandwidths is occupied in both directions. Then another constraint is generated:

$$n_{ij}^{v} = max(n_{ij}, n_{ji}) \tag{3.11}$$

The complete formulation is depicted in Table3.1. The implementation and solved results with linear programming solver tool CPLEX [28] will be discussed later.

3.3 Second Phase: HNN Based Routing Optimization

The reminder of this chapter focuses on developing the HNN model for our problem. We already know the HNN method is a very effective technique for solving constraint optimization problems [18] based on the Lyapunov energy function. A comparison between HNN and other optimization methods, including Genetic Algorithm (GA) and LP can be found in [20] where it has been proven that the HNN is the best method for extended optimization since it can reach convergence in the shortest time based on preliminary results. Therefore, we exploit HNN method in the second phase of our further optimization of the network performance. After providing a brief description of the conceptual structure for our HNN problem, we present the basic energy function and equation of motion that we have developed.

3.3.1 Neuron and Model

A Hopfield Neural Network (HNN) consists of a large amount of neurons that behave like simple analog amplifiers. In a HNN modeled system for optimization problems, the neurons can be used to represent variables with the value between 0 and 1. The outputs of them are approaching binary values when the neural network reaches an equilibrium state with a convergence of the energy function, which is called "minimum". The neuron model structure for a typical network is showed in Figure 3.6. Usually a neuron has several vectors: input, output and bias.

Table 3. 1 *Problem Formula*

Objective:

$$maximize \quad \left\{ \sum_{(s,d)} \sum_{\gamma \in R(s,d)} \delta^{v}_{sd} F^{v}_{(sd)\gamma} \right\} \tag{1}$$

Subject to:

-Flow conservation constraint

$$\sum_{\gamma \in R(s,d)} F^{v}_{(sd)\gamma} \le \alpha T_{sd} \tag{2}$$

-Capacity constraint

$$\sum_{(s,d)} \sum_{\gamma \in R(s,d)} F^{v}_{(sd)\gamma} \beta_{(sd)\gamma(ij)} + \frac{P}{del_{ij}} \le C_u n_{ij} \tag{3}$$

Objective:

$$max \left\{ max \sum_{(s,d)} \sum_{\gamma \in R(s,d)} \delta^{v}_{sd} F^{v}_{(sd)\gamma} + \sum_{(s,d)} \sum_{(i,j) \in L(s,d)} \delta^{d}_{sd} F^{d}_{ij} \tau^{d}_{ij} (1 - \rho^{d}_{sd}) - \sum_{(i,j) \in L} k \, (n^{v}_{ij} + n^{d}_{ij}) \right\} \tag{4}$$

Subject to:

-Flow conservation constraint

$$\sum_{i \in N | (i,j) \in L(s,d)} F^{d}_{ij} - \sum_{i \in N | (j,i) \in L(s,d)} F^{d}_{ji} = \begin{cases} (1-\alpha) T_{sd} & i = sour \\ -(1-\alpha) T_{sd} & i = dest \\ 0 & otherwise \end{cases} \tag{5}$$

-Capacity constraint

$$\sum_{(s,d)} \sum_{\gamma \in R(s,d)} F^{v}_{(sd)\gamma} \beta_{(sd)\gamma(ij)} + \sum_{(s,d):(i,j) \in L(s,d)} F^{d}_{ij} + \frac{P}{del_{ij}} \le C_u (n^{v}_{ij} + n^{d}_{ij}) \tag{6}$$

-Facilities and non-negativity constraints:

$$n^{v}_{ij} = max(n_{ij}, n_{ji}) \tag{7}$$

$$\tau^{d}_{ij} = \begin{cases} 1, & i = source \\ 0, & otherwise \end{cases} \tag{8}$$

$$n^{v}_{ij} + n^{d}_{ij} \le C_{ij} / C_u \tag{9}$$

$$\beta_{(sd)\gamma(ij)} = \begin{cases} 1, & \gamma th\ rout\ \ cross\ \ ijth\ \ link \\ 0, & otherwise \end{cases} \tag{10}$$

$$F^{v}_{(sd)\gamma} \ge 0 \tag{11}$$

$$F^{d}_{ij} \ge 0 \tag{12}$$

$$0 \le \alpha \le 1 \tag{13}$$

$$0 \le \rho^{d}_{sd} \le 1 \tag{14}$$

$$k \ge 0 \tag{15}$$

$$C_u \ge 0 \tag{16}$$

$$n^{v}_{ij} \ge 0 \tag{17}$$

$$n^{d}_{ij} \ge 0 \tag{18}$$

$$n_{ij} \ge 0 \tag{19}$$

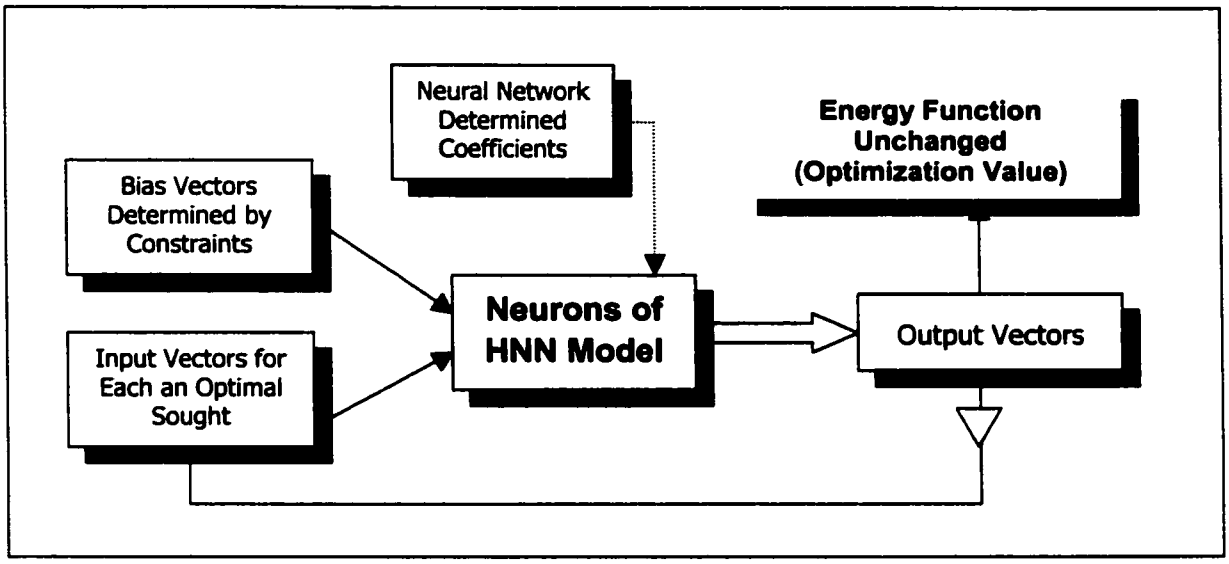

Figure 3. 6 *HNN Neuron-Model*

Based on Jeffrey E. Wieselthier and Anthony Ephremides's study [26], in our model, one neuron is defined to be the proportion of the traffic transmitted over each path between every SD pair. For example, Figure 3.7 shows the proportion-neuron model for a simple four-node network system. A double index is used to specify the neurons.

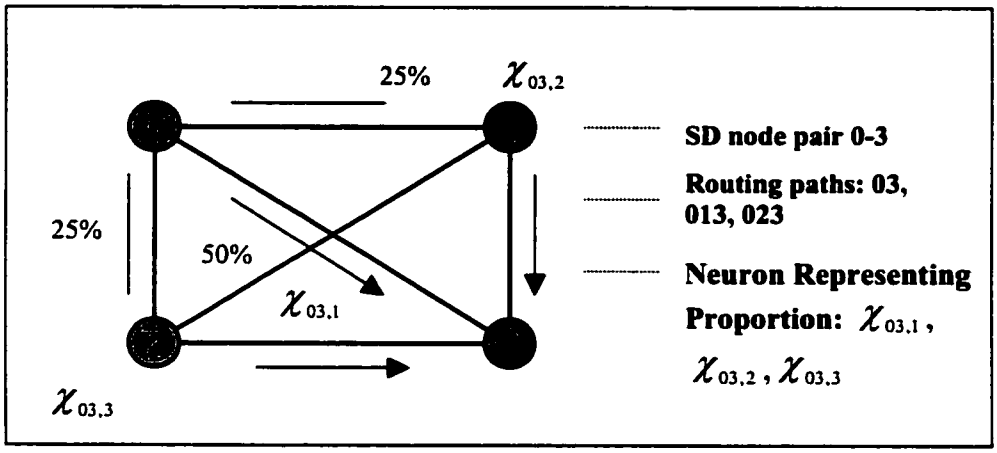

Figure 3. 7 *Proportion-Neuron Model*

With such definition above, introducing optimal picked-up routes, the extended calculation of the propagation average delay can be carried out in order to get the optimal

packet routing distribution. The following applications are executed with these two important steps:

1) Formulating the problem as the energy function
2) Finding the equation of motion for analogous approximation

Hereinafter we will discuss this solution step by step.

3.3.2 Energy Function

In this step, we are required to construct an energy function, which can reflect our objective as well as the requirement constraints. In general, such an energy function can be described in the form:

$$E = A \cdot (objective) + B \cdot (constraint\ 1) + \frac{C}{2} \cdot (constraint\ 2)$$

All the coefficients A, B, C in the equation are positive constants.

The voice service is supposed to be guaranteed with the lowest delay and is optimized separately in first phase, while the data service is a throughput-sensitive service and only uses the residual capacity for transmission in our problem. Moreover data services work well with discontinuous packetized transmission and aim at the highest throughput in contract to the voice packets [4]. Therefore it is reasonable for us to focus on the throughput of data services in this stage. So now our problem is, based on the optimal results of the first phase, to get an optimal status for the network performance of the data service with approaching a maximal throughput.

As aforementioned, OSPF is selected to be the basic routing protocol in our network system. This protocol is an interior gateway protocol used to distribute routing information with the following advantages [29]:

1) No limitation on the hop count.

2) IP multicast is used to send link-state updates, which ensures less processing on routers that are not listening to OSPF packets.

3) Updates are only sent in case routing changes occur instead of periodically. This ensures a better use of bandwidth and better convergence.

4) Allows for better load balancing.

Thus in some aspects, the throughput can be estimated by the amount of the propagation traffic flow staying in the whole network system. In order to get the highest throughput, the fast paths are always picked up to make the traffic leave the network as soon as possible. Hence we can easily get a substitution expression with the total traffic flow staying in the system as the following form:

$$F = \sum_{(s,d)} \sum_{\gamma \in R(s,d)} \sum_{l \in L} F_{sd} \chi_{sd\gamma} \beta_{(sd)\gamma l} \tag{3.12}$$

with the notation defined:

F_{sd}: total data traffic flow between (s,d) (kbps) which is obtained from the first phase optimization.

$\chi_{sd\gamma}$: the proportion of traffic between (s,d) transferred through the γth route.

l: the lth link.

$\beta_{(sd)\gamma l}$: if the γth route between (s,d) includes the lth link, then equals 1, else equals 0.

$R(s,d)$: the number of predetermined routes between (s,d).

Furthermore, we set each of the selected routes to correspond to a neuron of the HNN model and the variable $\chi_{sd\gamma}$ to be the output of the neuron because we need to compute

the proportion of the traffic carried through each routes. Equation 3.12 is used to be the "cost" of the total throughput. Before the complete mathematical formulations of the model are summarized, the following notation is defined:

P : average packet size (bit).

ad_l : maximum acceptable average delay of the *lth* link (s).

C_l : capacity of the *lth* link (kbps), also obtained from the first phase optimization.

Then a practical energy function can be constructed

$$E = AE_1 + BE_2 + CE_3 \tag{3.13}$$

1) ***Objective function***: Propagation traffic flow

$$E_1 = \sum_{(s,d)} \sum_{\gamma \in R(s,d)} \sum_{l \in L} F_{sd} \chi_{sd\gamma} \beta_{(sd)\gamma l} \tag{3.14}$$

As will be shown in the next chapter, this substitution does not prevent us from obtaining good solution.

2) ***Constraint1***: Activate exactly 100% of traffic flow per (s,d) pair

$$E_2 = \sum_{(s,d)} (\sum_{\gamma \in R(s,d)} \chi_{sd\gamma} - 1)^2 \tag{3.15}$$

This term subjects to the constraint of the neuron state $\chi_{sd\gamma}$. Because of the characteristics of the neuron, $0 \le \chi_{sd\gamma} \le 1$ is automatically satisfied. Moreover, we define $\chi_{sd\gamma}$ by the meaning of the proportion of the traffic, so that the equation $\sum_{\gamma} \chi_{sd\gamma} = 1$ should also be satisfied. The condition $E_2 = 0$ for the minimum energy function exactly guarantees this requirement.

3) ***Constraint2***: Activate no more traffic than the installed capacity of each link

$$E_3 = \sum_{l \in L} f^2 \left(\sum_{(s,d)\gamma \in R(s,d)} \sum F_{sd} \chi_{sd\gamma} \beta_{(sd)\gamma l} + \frac{P}{ad_l} - c_l \right), \qquad (3.16)$$

$$, where \ f(x) = \begin{cases} x, & if \ x \geq 0 \\ 0, & if \ x < 0 \end{cases}.$$

The capacity constraint is guaranteed by this term. Similar to the second term, only when $E_3 = 0$, the energy function reaches the minimum. Hence the following requirement is surely satisfied:

$$\sum_{(s,d)\gamma \in R(s,d)} \sum F_{sd} \chi_{sd\gamma} \beta_{(sd)\gamma l} + \frac{P}{ad_l} \leq c_l \qquad (3.17)$$

3.3.3 Equation of Motion

Obviously the neuron states are not binary values in our problem. So we cannot declare the neuron simply with the highest or the lowest state. An alternative way to gain the admissible solution is to define the evolution of the network following the original definition of the analogous Hopfield network, which is to determine a differential equation for each unit from the energy function in equation (3.13). The input $u_{sd\gamma}$ to a unit $sd\gamma$ is then characterized by the equation

$$du_{sd\gamma} = \left[-\frac{u_{sd\gamma}}{\tau} - A \sum_{l \in L} F_{sd} \beta_{(sd)\gamma l} - 2B \left(\sum_{\gamma l} \chi_{sd\gamma} - 1 \right) \right. \qquad (3.18)$$
$$\left. - 2C \sum_{l \in L} f \left(\sum_{(s,d)\gamma \in R(s,d)} \sum F_{sd} \chi_{(sd)\gamma} \beta_{(sd)\gamma l} + \frac{P}{ad_l} - c_l \right) \right] dt$$

where the time constant of the amplifiers in the system τ can be set to 1.0 with no loss of generality. The energy function will decrease monotonously until reach the stable equilibrium when the system evolution begins with a set of initial states.

Generally for each neuron, its action function typically uses the sigmoid function like

$$u_{sdy} = \frac{1}{2}(1 + \tanh(\frac{u_{sdy}}{u_0})) \tag{3.19}$$

as shown in Figure2.3. Here, u_0 is represents the slope of the nonlinear relation between the input and the output. In Ref [18], the time interval taken in the differential form of the above equation is purposed to be a fraction of τ. When the energy function decreases monotonously until the stable equilibrium is reached, equation (3.18) equals zero. Thus the updating rule of the evolution may be expressed in iterative form as follow:

$$u_{sdy}(t+1) = u_{sdy}(t) + \frac{d u_{sdy}}{dt}\delta t \tag{3.20}$$

3.3.4 Evolution Algorithm

The evolution of finding a solution is done by this implementing equation (3.19). Here we let the network evolve synchronously. In detail, it follows the iteration algorithm in Figure 3.8. When the deviation of the energy function becomes zero or below a given threshold, the energy function is said to be convergence, and an optimal solution is reached. A key issue is the ability of the convergence to a good solution strongly depending on the parameters in the above equations. The coefficients of the energy function, the time step and the slope parameter determine the speed and the results of the update, especially the values of coefficients have a great impact of the quality of the solution and the speed of convergence. Additionally, the time step for each of the iteration should be small enough.

These will be discussed detailed in next chapter when we introduce our software solution tools.

```
/* HNN main iteration algorithm implementation
 */
   set initial input [χ_xdy] and add perturbation;
/* update iteration for each neuron */
   do {
      compute output u_xdy(t+1) follows equation of motion;
      compute [χ'_xdy] follows action function;
      replace [χ_xdy] with [χ'_xdy];
      compute energy function;    /* for check */
   } while u_xdy(t+1)=u_xdy(t) or d u_xdy ≤ threshold
/* each step decreasing the energy function quality and
 * convergence to a fixed point is assured to get the optimum
 */
    compute objective;
```

Figure 3. 8 ***HNN Iteration Algorithm***

Chapter 4

Numerical Results and Analysis

This chapter describes the details of the proposed two-phase design we have introduced in the previous chapters, and report numerical results. Both the models have been applied to a nation-wide Internet backbone network. The proposed models are implemented in C and Java, and Linear Programming (LP) tool CPLEX. The aim of this implementation exercise is to validate the abstract design method, to gain the optimal design for the network system, and further to get insight into the performance aspects of the models.

4.1 Infrastructure of Colombia Nation-wide Internet Backbone

4.1.1 Infrastructure

The infrastructure on which study and simulation were performed is a simplified Internet backbone system based on the current network of Colombia. Twenty cities have been picked up among 52 cities all over the country, nine of which are main cities with population more than 120,000 and have most frequent data transmission. The observed network has nine nodes $(N = 9)$, of which 14 pairs of nodes directly connected as shown in Figure 4.1. In this thesis, we define a directed link as a data connection with a unique <source→destination> tuple. Thus 28 links $(L = 2 \times 14)$ will be calculated. The colors of the links, called weight, represent the different bandwidths. According to the statistic data of Colombia Telecommunication, all the International traffic traverses the Capital Bogota. The three largest cities Bogota, Medellín and Cali are directly connected with

each other to form a widest bandwidth ring, while the other six cities connected to the ring and to be the backbone network with it together. The communications between these smaller-size cities have to traverse one of the largest three cities to reach the destinations except there are direct links between them. In addition, the small cities, the populations of which are lower than 120,000, are represented by blue circles. Almost all the traffic of these cities is downstream and very low. No admission routing problem is concerned in these nodes and just treated as subscribers of those bigger cities. Table 4.1 gives the list of all the directed links and their installed capacities.

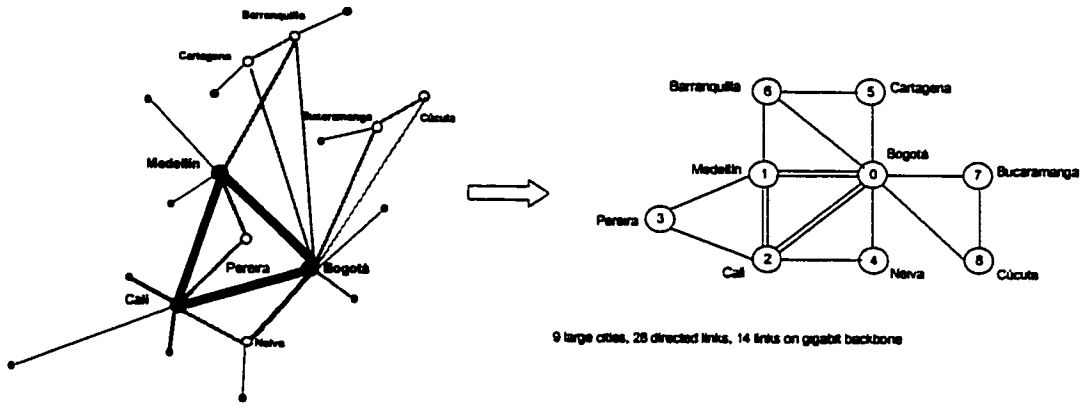

Figure 4. 1 ***Infrastructure of Colombian Nation-Wide Backbone***

4.1.2 Traffic Matrix and Multiplier

Based on the aforementioned assumptions, two service classes are considered, of which, the voice service is characterized by real-time constraints while data service is a delay-insensitive service. To be convenient, a base traffic demand matrix ($T(T_{sd})$) and a multiplier (α) for voice service are applied. The traffic demands between each of the 31 (s,d) pairs are given in Table 4.2.

Table 4. 1 *Installed Capacity*

No	i-j	Capacity	No	i-j	Capacity
0	0-1	3398540	14	2-4	614400
1	0-2	2125068	15	3-1	60000
2	0-4	621400	16	3-2	312148
3	0-5	51538	17	4-0	621400
4	0-6	131264	18	4-2	614400
5	0-7	751852	19	5-0	51538
6	0-8	15424	20	5-6	54148
7	1-0	3398540	21	6-0	131264
8	1-2	3359864	22	6-1	311000
9	1-3	60000	23	6-5	54148
10	1-6	311000	24	7-0	751852
11	2-0	2125068	25	7-8	29348
12	2-1	3359864	26	8-0	15424
13	2-3	312148	27	8-7	29348

Table 4. 2 *Traffic Demand Matrix*

SD	0	1	2	3	4	5	6	7	8
0	-	359278.8	224652.9	4127.98	65697.7	5448.37	13876.66	79482.51	999.55
1	239086.8	-	73821.2	1318.29	0	0	0	0	0
2	93010.1	26558.3	-	474.27	4856.6	0	0	0	0
3	8604.5	594.5	3093.1	-	0	0	0	0	0
4	268.2	0	76.4	38.4	-	0	0	0	0
5	13306.4	0	0	0	0	-	0	5702.76	0
6	18660.8	3405.8	0	0	0	592.97	-	0	0
7	50858.0	0	0	0	0	0	0	-	400.02
8	14631.8	0	0	0	0	0	6833.13	6270.76	-

Table 4. 3 *Preliminary Routing Table*

No	Sour-Dest	Routing Paths (no more than 3 intermediate hops)
0	0-1	0-1, 0-2-1, 0-2-3-1, 0-4-2-1, 0-4-2-3-1, 0-6-1, 0-5-6-1
1	0-2	0-1-2, 0-1-3-2, 0-2, 0-4-2, 0-5-6-1-3, 0-6-1-2, 0-6-1-3-2
2	0-3	0-1-3, 0-2-3, 0-4-2-3, 0-5-6-1-3, 0-6-1-3
3	0-4	0-1-3-2-4, 0-2-4, 0-4, 0-6-1-2-4
4	0-5	0-1-6-5, 0-2-1-6-5, 0-5, 0-6-5
5	0-6	0-1-6, 0-2-1-6, 0-2-3-1-6, 0-4-2-1-6, 0-5-6, 0-6
6	0-7	0-7, 0-8-7
7	0-8	0-7-8, 0-8
8	1-0	1-0, 1-2-0, 1-2-3-4-0, 1-2-4-0, 1-3-2-0, 1-6-0, 1-6-5-0
9	1-2	1-0-2, 1-0-4-2, 1-2, 1-3-2, 1-6-0-2, 1-6-0-4-2, 1-6-5-0-2
10	1-3	1-0-2-3, 1-0-4-2-3, 1-2-3, 1-3, 1-6-0-2-3
11	2-0	2-0, 2-1-0, 2-1-6-0, 2-1-6-5-0, 2-3-1-0, 2-3-1-6-0, 2-4-0
12	2-1	2-0-1, 2-0-6-1, 2-0-5-6-1, 2-1, 2-3-1, 2-4-0-1, 2-4-0-6-1
13	2-3	2-0-1-3, 2-0-6-1-3, 2-1-3, 2-3, 2-4-0-1-3
14	2-4	2-0-4, 2-1-0-4, 2-1-6-0-4, 2-3-1-0-4, 2-4
15	3-0	3-1-0, 3-1-6-0, 3-1-6-5-0, 3-2-0, 3-2-4-0
16	3-1	3-1, 3-2-1, 3-2-0-1, 3-2-0-6-1, 3-2-4-0-1
17	3-2	3-1-0-2, 3-1-0-4-2, 3-1-2, 3-1-6-0-2, 3-2
18	4-0	4-0, 4-2-0, 4-2-1-0, 4-2-1-6-0, 4-2-3-1-0
19	4-2	4-0-2, 4-0-1-2, 4-0-6-1-2, 4-0-1-3-2, 4-2
20	4-3	4-0-1-3, 4-0-6-1-3, 4-2-3
21	5-0	5-0, 5-6-0, 5-6-1-0, 5-6-1-2-0
22	5-7	5-0-7, 5-6-0-7, 5-6-1-0-7
23	6-0	6-0, 6-1-0, 6-1-2-0, 6-1-2-4-0, 6-1-3-2-0, 6-5-0
24	6-1	6-0-1, 6-0-2-1, 6-0-2-3-1, 6-0-4-2-1, 6-1, 6-5-0-1
25	6-5	6-0-5, 6-1-0-5, 6-1-2-0-5, 6-5
26	7-0	7-0, 7-8-0
27	7-8	7-0-8, 7-8
28	8-0	8-0, 8-7-0
29	8-6	8-0-1-6, 8-0-2-1-6, 8-0-5-6, 8-0-6, 8-7-0-1-6, 8-7-0-5-6, 8-7-0-6
30	8-7	8-0-7, 8-7

4.1.3 Algorithm for Routing Path Selection

The preliminary routing paths listed in Table 4.3, including the 146 paths for voice service admission routing in LP optimization, is generated by finding all the routes between each (s,d) pair crossing no more than 3 intermediate nodes. The routing paths

using in the HNN optimization for data service were generated via an algorithm that finds at most 4 routes crossing least same nodes and less than h hops. This algorithm is shown in Figure 4.2. In this algorithm, (s,d), a pair of nodes, is used as a parameter of the function. At the beginning of the function, each node and each arc are initialized by giving each of them a weight which is equal to 1 in order to indicate the length of the routing path. By repeating the utilization of the Dijkstra's shortest-path algorithm (see Appendix C), the shortest path between each (s,d) can be always selected. Here the distance of the path is indicated by the summation of the nodes crossed by it. After this, a checksum Hop[path] is calculated to guarantee the intermediate hop limitation. The update of the node marks helps the each loop to get a new "shortest" path with the lowest probability of traversing the same nodes. Thus all $R(s,d)$ can be found by calling "Function (s,d)" repeatedly. All the paths are no more than the limited hops. A set of 31 (s,d) pairs and 110 final selected routing paths are enumerated in Table 4.9.

```
Function Rout_pick (s,d) {
   mark each node with 1;
   mark each arc with 1;
   do   {
      get the path having minimum summation of nodes
      between(s,d) using Dijkstra's shortest-path algorithm;
      add path to set R(s,d);
      Hop[path]=sum of the arcs crossed by the path;
      update the mark of each nodes in the path with n;
   } while ((Hop[path] <h) || (less than 4 routes));
   write R(s,d);
}
```

Figure 4. 2 *Algorithm of Routing Selection*

4.2 Results - LP Method

4.2.1 Program Structure

The first LP model for optimizing the Internet backbone system described in last subsection is implemented using the CPLEX callable library of C. The whole workspace includes three main files for each objective:

- coltest.c: This file is used to pass in the data defining the model, and place the nodal characteristics into the *lp* object already created by the caller. The result of the routine is that the object *lp* is modified to hold the entire model. The GenNt function generates model by creating the matrix coefficients by column.
- testdrv.c: This file contains the driver function to solve the model of network optimization in the first phase. The data is defined at compile time, and the routine *nt_opt*_and_*soln* is called to generate and optimize the problem, returning the part of the solution we are interested in.
- testsub.c: This file contains a function to take as input the parameters defining the network optimal model, and total profit as output.

4.2.3 Results

The parameter values listed in Table 4.4 are used for our examination, which also may be used to tune the performance of the results.

Table 4. 4 *Parameter Values Used in Simulations*

k	δ^{v}_{sd}	δ^{d}_{sd}	α	del_{ij}	C_u	P	ρ^{d}_{sd}
1.0	1.0	1.0	0.4	60ms	155kbps	400bytes	1%

The results of LP model are given in three parts: first, we give the maximum network revenue for the voice service and its profit; second, we present the maximum total network profit including both service classes; and at last, we list the exact carried bandwidth and the capacity finally assigned for the data service, which are the inputs in the HNN model.

As we have already mentioned in last section, the voice service is given a higher priority in using the entire installed capacity. So the foremost observation on the results is that almost no loss of the voice service with a given limit average delay. Table 4.6 reports the results from the voice LP model and the traffic distribution is presented in Table 4.7. All the voice traffic select one route to departure, and most of which are the shortest one except the traffic stream between node pair 29 -from Cúcuta 8 to Barranquilla 6. This stream travels through the route 8-7-0-1-6 but not the shortest one 8-0-6. From the usage percentage of each link, we can see the load on link 8-0 is highest while the capacity of this link is the lowest of the all 28 links. So the routes 8-7-0-1-6 will be the better selection.

Table 4. 5 *Results of Voice Service*

Objective value (Profit)	526923
Solution (Revenue)	optimal
Total voice demand	530411.7
Total voice bandwidth carried	530411.49
Loss	3.95E-07

After optimizing the voice service revenue, with the residual capacities being the constraint, we calculate the total optimal profit value and check the whole Internet working states. Table 4.8 reports the combination optimal results, and Figure 4.3 and 4.4 present the link capacity and the traffic traveling over each link.

Table 4. 6 *Voice Traffic Distribution*

sour-dest	Rout0	Rout1	Rout2	Rout3	Rout4	Rout5	Rout6	Carried Bandwidth	Selected Routes	Total Traffic Demand
0	143712	0	0	0	0	0	0	143712	0-1	359278.8
1	0	0	89861.2	0	0	0	0	89861.2	0-2	224652.9
2	1651.19	0	0	0	0			1651.19	0-1-3	4127.98
3	0	0	26279.1	0				26279.1	0-4	65697.7
4	0	0	2179.35	0				2179.35	0-5	5448.37
5	0	0	0	0	0	5550.66		5550.66	0-6	13876.66
6	31793	0						31793	0-7	79482.59
7	0	399.82						399.82	0-8	999.55
8	95634.7	0	0	0	0	0	0	95634.7	1-0	239086.8
9	0	0	29528.5	0	0	0	0	29528.5	1-2	73821.2
10	0	0	0	527.316	0			527.316	1-3	1318.29
11	37204	0	0	0	0	0	0	37204	2-0	93010.9
12	0	0	0	10623.3	0	0	0	10623.3	2-1	26558.3
13	0	0	0	189.708	0			189.708	2-3	474.27
14	0	0	0	0	1942.64			1942.64	2-4	4856.6
15	3441.8	0	0	0	0			3441.8	3-1-0	8604.5
16	237.8	0	0	0	0			237.8	3-1	594.5
17	0	0	0	0	1237.24			1237.24	3-2	3093.1
18	107.28	0	0	0	0			107.28	4-0	268.2
19	0	0	0	0	30.56			30.56	4-2	76.4
20	0	0	15.36					15.36	4-2-3	38.4
21	5322.56	0	0	0				5322.56	5-0	13306.4
22	2281.1	0	0					2281.1	5-0-7	5702.76
23	7464.32	0	0	0	0	0		7464.32	6-0	18660.8
24	0	0	0	0	1362.32	0		1362.32	6-1	3405.8
25	0	0	0	237.188				237.188	6-5	592.97
26	20343.2	0						20343.2	7-0	50858
27	0	160.008						160.008	7-8	400.02
28	5852.72	0						5852.72	8-0	14631.8
29	0	0	0	0	2733.25	0	0	2733.25	8-7-0-1-6	6833.93
30	0	2508.3						2508.3	8-7	6270.76
Total								530411.49		1326029.25
Loss								3.95919E-07		

Table 4. 7 *Result of Both Service Classes*

Objective Value (Profit)	1309303
Solution	optimal
Voice Profit	526923
Data Profit	782380
Total bandwidth Carried for Data	783074.95

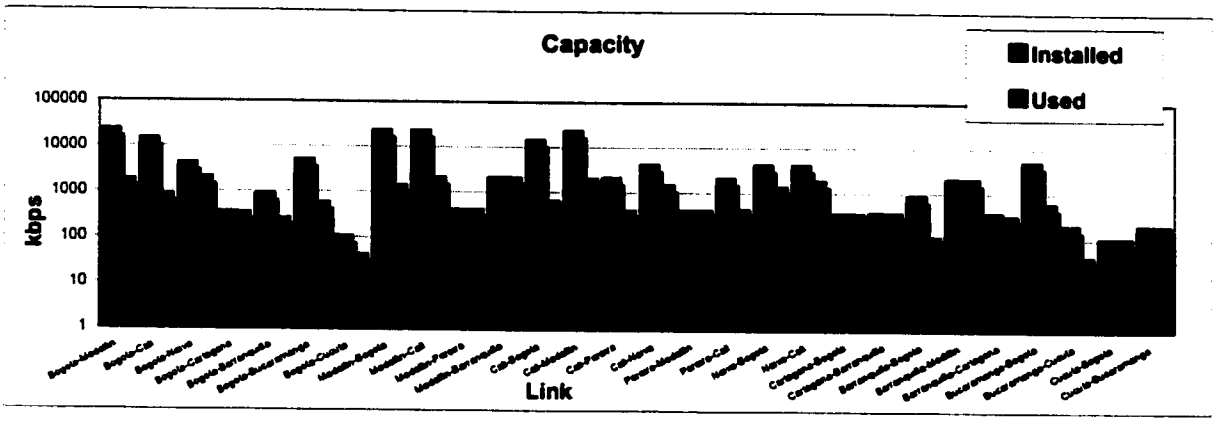

Figure 4. 3 *Comparison of Installed Capacity and Used Capacity*

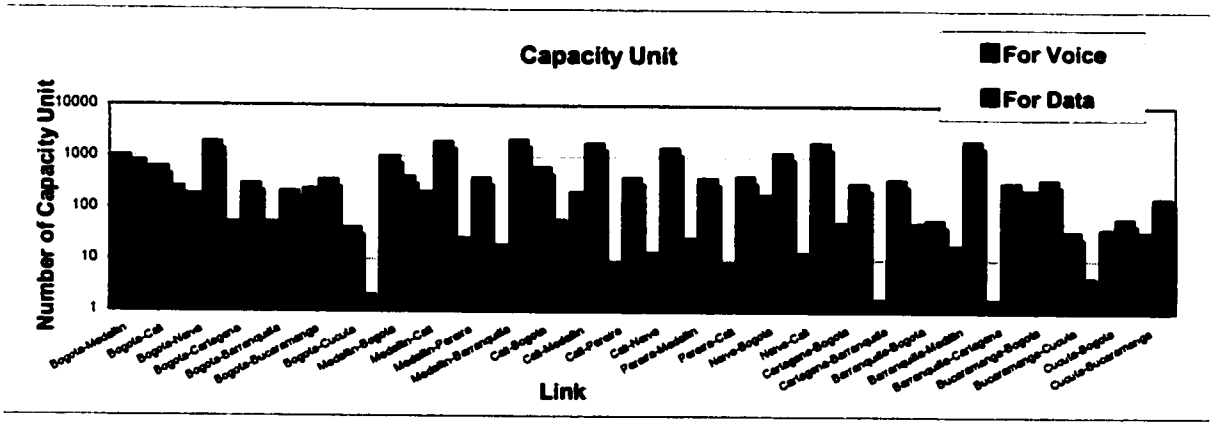

Figure 4. 4 *Comparison of Capacity for Voice and Data*

The performance of data streams is quite different from the voice traffic. Several of them are split into more than one route as shown in the following table:

Table 4. 8 *Split Data Streams*

S-D	Paths
0-2	0-1-2, 0-1-3-2, 0-5-6-1-2, 0-6-1-2
1-2	1-0-4-2, 1-6-5-0-4-2
2-0	2-1-0, 2-3-1-0
8-6	8-0-2-1-6, 8-7-0-2-1-6

From the LP model, an optimized profit for the multi-service Internet Backbone system is obtained. Meanwhile, this simulation makes good traffic assignments and capacity usages for the voice service because it is given a higher priority than the data service. However the situation for the data service is different due to the lower priority level and the less consideration. There is more traffic loss happened, for instance on the path 8-0 and 8-7. In order to make the method more effective, we continue the optimization of the network performance for data services with the results we got.

4.3 Results - HNN Method

4.3.1 Program Structure

In this section, we will introduce our Hopfield net system, which is implemented in Java, for the optimization carried in the second phase. Figure 4.5 shows the organization of the significant classes for this system.

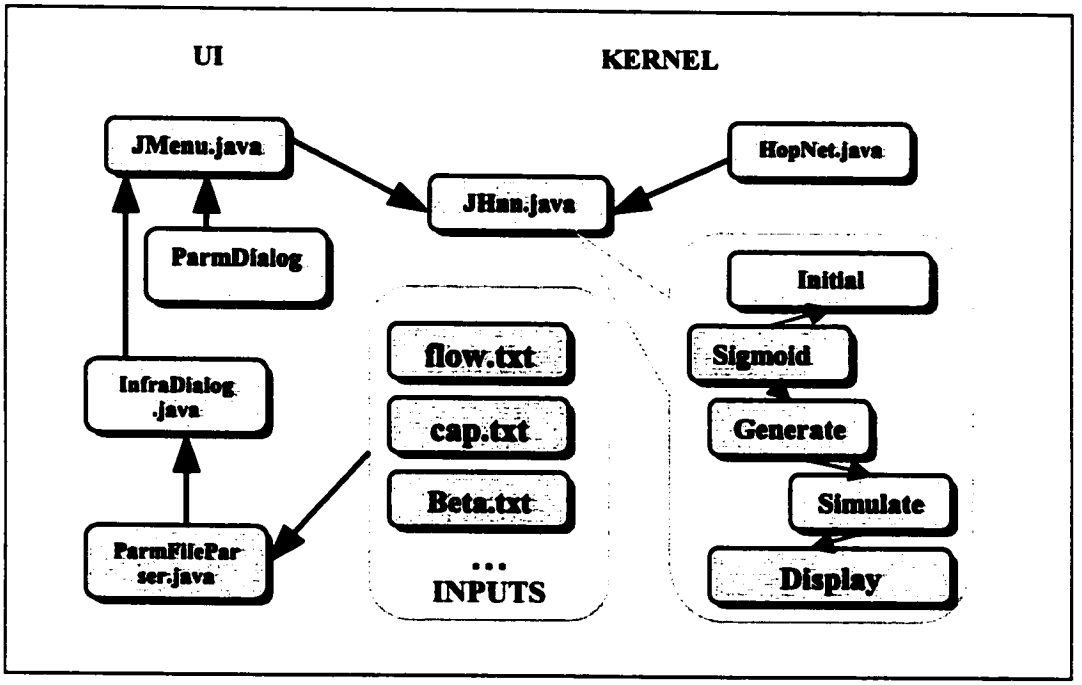

Figure 4. 5 *Program Organization*

There are two major components in the program: UI and KERNEL. The KERNEL is responsible for simulating the neural network structure and the neuron behavior. The classes in UI are responsible for reading the constants, the parameters and the variables, which are common to the entire system. The system setup record to be supplied by the user interface contains parameters, coefficients and infrastructure setting. The UI (see Figure 4.6) has been designed to facilitate the management of the large number of different experiments. The UI can also be used as a generic HNN tool.

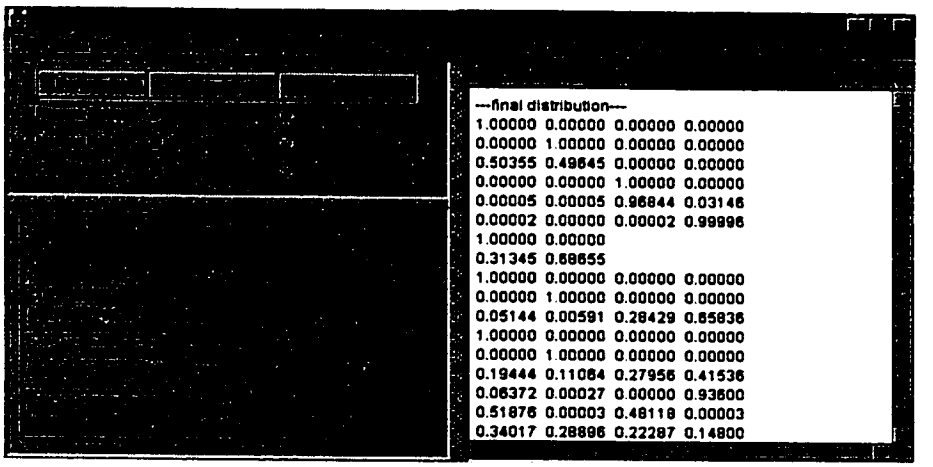

Figure 4. 6 *Toolkit Interface*

In this program, each neuron ("unit" in program) has four states: "act", "excit", "inhibit", and "bias". The first three handled the renewal of the neuron state and the last one is for constant bias of the neuron. We try two different common methods for the neuron initialization:

1. *Sigmoid Initialization* -

```
for (i=0;i<NPAIR;i++)

   for (j=0;j<ROUT[i];j++)  {
      array[row].excit=wght[i]*(1.0+Rand());
      array[row].act=logistic(array[row].excit, temp);
      row++;
}
```

2. *Average Initialization* -

```
for (i=0;i<NPAIR;i++)
    for (j=0;j<ROUT[i];j++)  {
       array[row].act=1*(1.0+Rand())/ ROUT[i];
       row++;
}
```

The differential of the energy function is calculated as following in each time step:

```
/* the second term */
ub += array[row1].act;
/* the first term */
ua = array[row].bias;
/* the third term */
dd[k]+=flow[m]*beta[row2][k]*array[row2].act;
uc +=f(dd[k]+P/ad-C[k]);
/* update the total input to a unit */
array[row].excit -=
dt*(array[row].excit+a*ua+2*b*(ub-1)+2*c*uc);
```

4.3.2 Coefficients and Input Files

Before running the simulation program, we should first determine the suitable values for coefficients in the energy function in order to reach the final steady states. In the following we present the generation approach used in our experiments which is based on the theory developed in Ref [32]:

First we assume among all the entries in the $(sd)th$ row, only $\chi_{(sd)\gamma_1}$ is larger than 0, while others are zeros. Thus,

$$\frac{du_{(sd)\gamma}}{dt} = -Aua - 2B(\chi_{(sd)\gamma_1} - 1) - 2Cuc \tag{4.1}$$

where

$$ua = \sum_l F_{sd}\beta_{(sd)\gamma l} \tag{4.2}$$

$$uc = \sum_l f(\sum_{(s,d)}\sum_{\gamma} F_{sd}\chi_{(sd)\gamma}\beta_{(sd)\gamma l} + \frac{P}{ad_l} - c_l) \tag{4.3}$$

$$\chi_{(sd)r_1} = 1 - \frac{Aua + 2Cuc}{2B} \geq 0 \tag{4.4}$$

Thus we obtain the first relation (4.5) between A, B, and C,

$$B \geq \frac{A*max(ua) + 2*C*max(uc)}{2} \tag{4.5}$$

Now we assume that $\chi_{i\gamma} > \chi_{j\gamma}$, hence when the network converges to a stable state, $\frac{du_{i\gamma}}{dt} < \frac{du_{j\gamma}}{dt}$ should be satisfied. With further assumption, there is only one entry for each column is greater than 0 except the γth column with two non-zero entries $\chi_{i\gamma}$ *and* $\chi_{j\gamma}$. By choosing $\chi_{(sd)r_1} = 1 \text{if}(sd) \neq i, j$, we generalize the following inequality,

$$-A\sum_l F_{sd}\beta_{i\gamma l} - 2B(\chi_{i\gamma} - 1) - 2Cuc < -A\sum_l F_{sd}\beta_{j\gamma l} - 2B(\chi_{j\gamma} - 1) - 2Cuc \tag{4.6}$$

Thus we can obtain another inequality,

$$B > \frac{A}{2}\sum_l F_{sd}(\beta_{i\gamma l} - \beta_{j\gamma l}) \tag{4.7}$$

Because the longest routing path is 3 hops, there are the following characters of $\beta_{(sd)\gamma l}$,

$$max\sum_l(\beta_{i\gamma l} - \beta_{j\gamma l}) = 5 \tag{4.8}$$

$$max(\sum_l \beta_{(sd)rl}) = 4 \tag{4.9}$$

Moreover, $max(uc) < max[\sum_{l} f(\sum_{(s,d)} F_{sd}\beta_{(sd)l} + \frac{P}{ad_l} - c_l)]$

$$< \sum_{l} f(5max F_{sd} + \frac{P}{ad_l} - c_l) \quad (4.10)$$

Therefore, from inequality (4.5) and (4.7), the final selected coefficient relation is in the following constraints:

$$B > 2A(max F_{sd}) + C\sum_{l} f(5max F_{sd} + \frac{P}{ad_l} - c_l) \quad (4.11)$$

$$A < \frac{2}{5} min(F_{sd}) \quad (4.12)$$

In our experiments, we let $A=1$, the values of C are between 0.01 and 1, and B is generated differently according the constraint (4.11) and (4.12).

Three input files include "flow.txt", "cap.txt" and "Beta.txt", all of which can be dynamically input through the menu. "flow.txt" is the matrix of the traffic flow between each (s,d) pair. "cap.txt" is the array of the capacity on each direct link. The first phase, we have already obtained the traffic flow and the capacity used by the data service. Since the data service revenue is only influenced by the amount of the departure streams, the reassignments of the same traffic demand between each (s,d) will not change the value of the revenue. Using these data makes it possible to keep the optimal network profit when we do the further optimal of the data throughput. "Beta.txt" is the set of indicator $\beta_{(sd)l}$.

As aforementioned, a predetermined routing table obtained with the algorithm (see Figure 4.2), is given to data packet transmission. Table 4.9 shows all the 110 routing paths, which generate the "Beta.txt" file.

Table 4. 9 *Routing Table for HNN Simulation*

No	Sour-Dest	Routing Paths			
0	0-1	0-1	0-2-1	0-4-2-1	0-6-1
1	0-2	0-1-2	0-2	0-4-2,	0-6-1-2
2	0-3	0-1-3	0-2-3	0-4-2-3	0-6-1-3
3	0-4	0-1-3-2-4	0-2-4	0-4	0-6-1-2-4
4	0-5	0-1-6-5	0-2-1-6-5	0-5	0-6-5
5	0-6	0-1-6	0-2-1-6	0-5-6	0-6
6	0-7	0-7	0-8-7		
7	0-8	0-7-8	0-8		
8	1-0	1-0	1-2-0	1-3-2-0	1-6-0
9	1-2	1-0-2	1-0-4-2	1-2	1-3-2
10	1-3	1-0-2-3	1-2-3	1-3	1-6-0-2-3
11	2-0	2-0	2-1-0	2-3-1-0	2-4-0
12	2-1	2-0-1	2-1	2-3-1	2-4-0-1
13	2-3	2-0-1-3	2-1-3	2-3	2-4-0-1-3
14	2-4	2-0-4	2-1-0-4	2-3-1-0-4	2-4
15	3-0	3-1-0	3-1-6-0	3-2-0	3-2-4-0
16	3-1	3-1	3-2-1	3-2-0-1	3-2-4-0-1
17	3-2	3-1-0-2	3-1-0-4-2	3-1-2	3-2
18	4-0	4-0	4-2-0	4-2-1-0	4-2-3-1-0
19	4-2	4-0-2	4-0-1-2	4-0-1-3-2	4-2
20	4-3	4-0-1-3	4-0-6-1-3	4-2-3	
21	5-0	5-0	5-6-0	5-6-1-0	5-6-1-2-0
22	5-7	5-0-7	5-6-0-7	5-6-1-0-7	
23	6-0	6-0	6-1-0	6-1-2-0	6-5-0
24	6-1	6-0-1	6-0-2-1	6-1	6-5-0-1
25	6-5	6-0-5	6-1-0-5	6-1-2-0-5	6-5
26	7-0	7-0	7-8-0		
27	7-8	7-0-8	7-8		
28	8-0	8-0	8-7-0		
29	8-6	8-0-1-6	8-0-6	8-7-0-1-6	8-7-0-6
30	8-7	8-0-7	8-7		

4.3.3 Results

By a large number of different selections of the values of A, B, C which satisfy the relationship above, different average packet sizes and corresponding acceptable average

delay, we did 250 experiments for each different initialization. Every experiment reached convergence and optimal. The parameter values listed in Table 4.10 are used for our examination, which also may be tuned to do comparisons.

Table 4. 10 ***Parameter Values Used in Simulation***

Packet size (bytes)	Average delay (ms)
400	60
1000	90

The following figures show the corresponding results of different initialization methods and parameters.

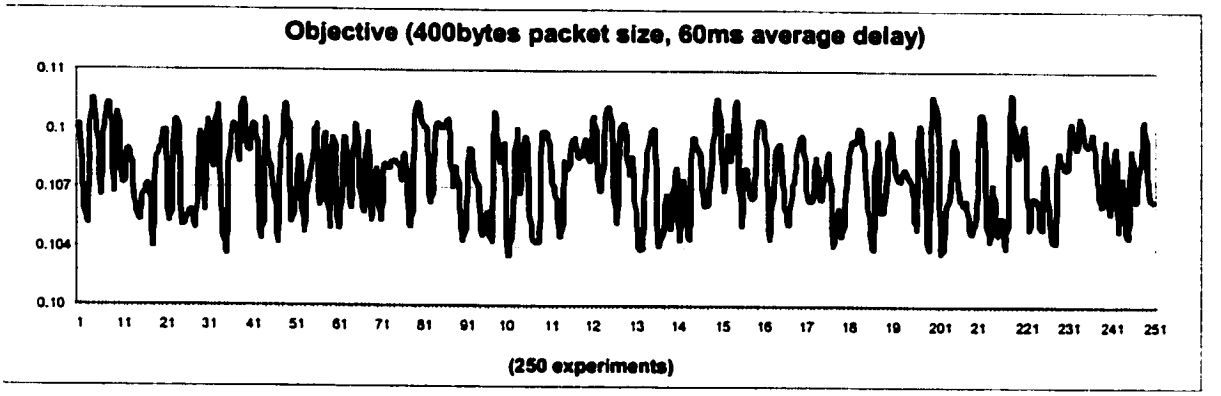

Figure 4. 7 ***Results of 400 bytes by Sigmoid Initialization***

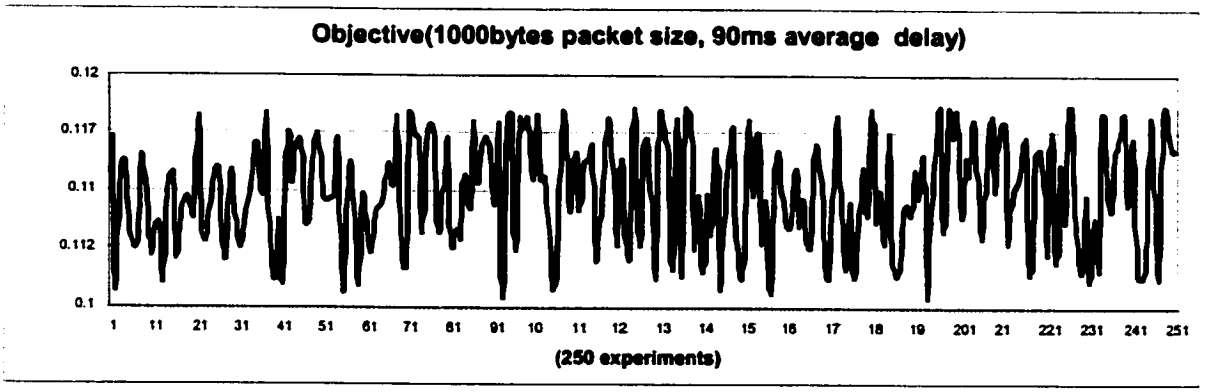

Figure 4. 8 ***Results of 1000 bytes by Sigmoid Initialization***

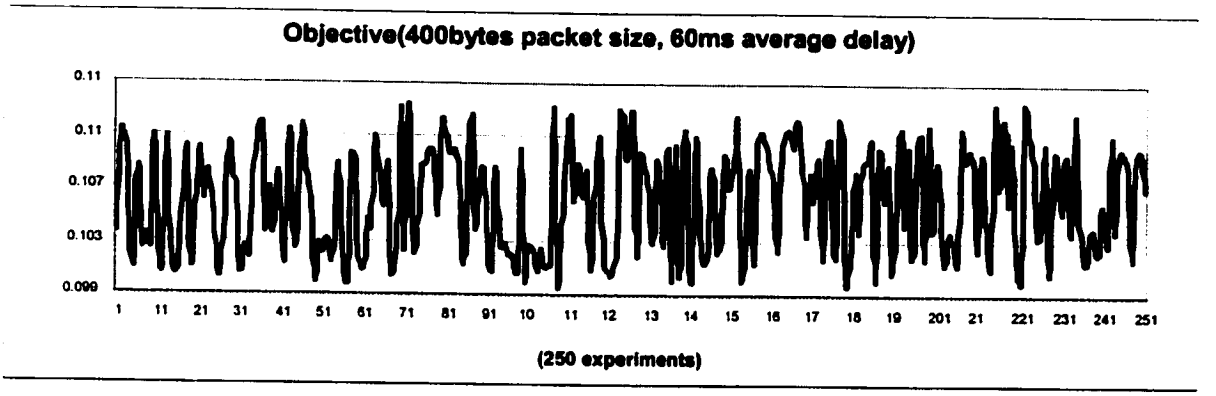

Figure 4. 9 ***Results of 400 bytes by Average Initialization***

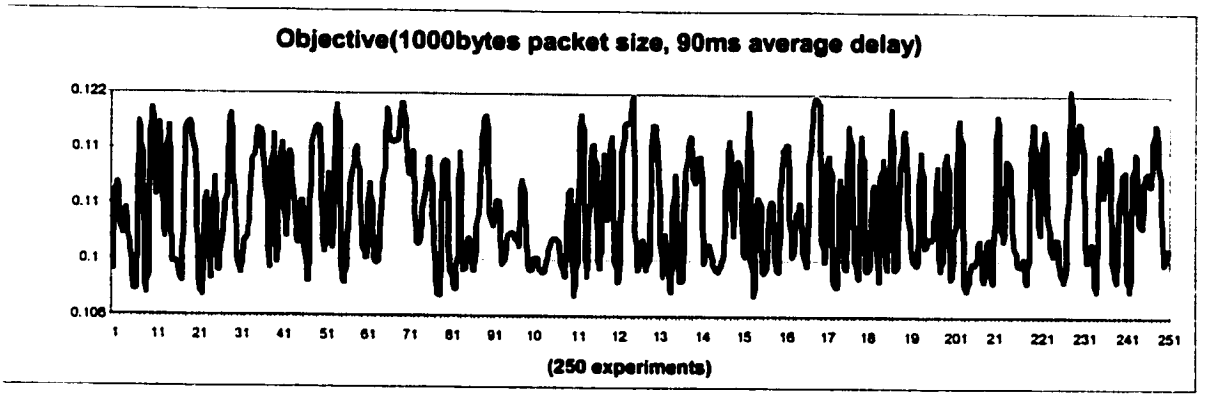

Figure 4. 10 ***Results of 1000 bytes by Average Initialization***

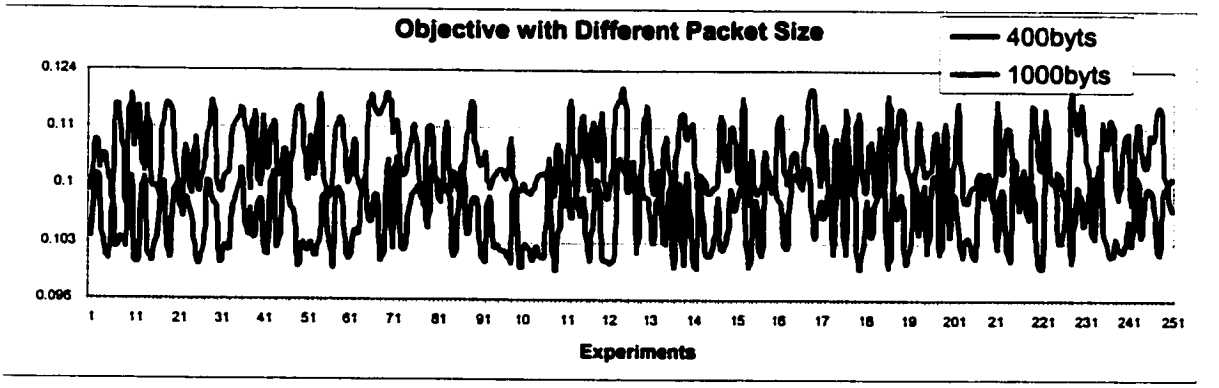

Figure 4. 11 ***Comparison of Different Packet Size***

Although the average initialization method leads to better objective value (see Table 4.12), the differential of this method, which is even larger than 1%, is much higher than sigmoid initialization.

Table 4. 11 ***Comparison of Different Initialization***

Objective	400 bytes, 60ms		1000 bytes, 90ms	
	Sigmoid	average	sigmoid	average
Average	0.1077318	0.1060804	0.1149301	0.11354729
Differential	0.00803	0.0119	0.00816	0.01456
Best case	0.10361	0.10005	0.11032	0.10763
Worst case	0.11164	0.11195	0.11848	0.12219

When the neural network converges at a stable state, we can finally get the selected path table for each optimal case. From all the 1000 experiments, the distributions of traffic between each SD pair are stable and similar routes are selected by the data streams. Table 4.12 is showing the different selections of the final traffic assignments for best case and worst case (the full comparison can be seen in Table 4.13). (Best case: 1000 bytes packet size, 90 ms average delay, and worst case: 400 bytes packet size, 60 ms average delay)

Table 4. 12 ***Different Selections between Best Case and Worst Case***

S-D	Best Case	Worst Case
0-5	0, 1, 2	2, 3
0-6	0, 3	3
2-3	0, 1, 2	0, 1, 2 , 3
2-4	0, 1, 3	0, 3
3-1	0, 1	0, 1, 2, 3
4-0	0, 1	0, 1, 2, 3
4-2	0, 1, 2	0, 1, 2, 3
6-5	0, 1	0, 1, 2, 3
8-6	0, 1, 2	0, 1, 2, 3

Table 4. 13 *Comparison of the Best Case and the Worst Case*

Source-Destination	Traffic Assign Proportion				Selected Routes (Best case)				Traffic Assign Proportion				Selected Routes (Worst case)			
	Rout0	Rout1	Rout2	Rout3	Rout0	Rout1	Rout2	Rout3	Rout0	Rout1	Rout2	Rout3	Rout0	Rout1	Rout2	Rout3
0-1	100.00%	0.00%	0.00%	0.00%	0-1				100.00%	0.00%	0.00%	0.00%	0-1			
0-2	0.00%	100.00%	0.00%	0.00%		0-2			0.00%	100.00%	0.00%	0.00%		0-2		
0-3	93.35%	6.65%	0.00%	0.00%	0-1-3	0-2-3			54.84%	45.16%	0.00%	0.00%	0-1-3	0-2-3		
0-4	0.00%	0.00%	100.00%	0.00%			0-4		0.00%	0.00%	100.00%	0.00%			0-4	
0-5	10.49%	0.13%	89.36%	0.01%	0-1-6-5	0-2-1-6-5	0-5		0.01%	0.01%	97.37%	2.61%			0-5	0-6-5
0-6	27.40%	0.00%	0.01%	72.59%	0-1-6			0-6	0.00%	0.00%	0.00%	99.99%				0-6
0-7	100.00%	0.00%			0-7				100.00%	0.00%			0-7			
0-8	84.62%	15.38%			0-7-8	0-8			40.83%	59.17%			0-7-8	0-8		
1-0	100.00%	0.00%	0.00%	0.00%	1-0				100.00%	0.00%	0.00%	0.00%	1-0			
1-2	0.00%	100.00%	0.00%	0.00%		1-0-4-2			0.00%	100.00%	0.00%	0.00%		1-0-4-2		
1-3	95.52%	2.20%	2.07%	0.21%	1-0-2-3	1-2-3	1-3	1-6-0-2-3	6.97%	0.58%	24.81%	67.64%	1-0-2-3	1-2-3	1-3	1-6-0-2-3
2-0	100.00%	0.00%	0.00%	0.00%	2-0				100.00%	0.00%	0.00%	0.00%	2-0			
2-1	0.00%	100.00%	0.00%	0.00%		2-1			0.00%	100.00%	0.00%	0.00%		2-1		
2-3	92.53%	7.20%	0.27%	0.01%	2-0-1-3	2-1-3	2-3		28.54%	16.43%	26.16%	28.87%	2-0-1-3	2-1-3	2-3	2-4-0-1-3
2-4	98.46%	0.19%	0.00%	1.36%	2-0-4	2-1-0-4		2-4	9.58%	0.03%	0.00%	90.39%	2-0-4			2-4
3-0	99.46%	0.00%	0.54%	0.00%	3-1-0		3-2-0		59.03%	0.01%	40.97%	0.00%	3-1-0		3-2-0	
3-1	92.90%	7.06%	0.04%	0.00%	3-1	3-2-1			48.09%	30.47%	15.13%	6.32%	3-1	3-2-1	3-2-0-1	3-2-4-0-1
3-2	64.27%	35.55%	0.00%	0.18%	3-1-0-2	3-1-0-4-2		3-2	1.37%	20.77%	0.03%	77.84%	3-1-0-2	3-1-0-4-2		3-2
4-0	91.80%	8.13%	0.07%	0.00%	4-0	4-2-0			39.41%	29.39%	19.51%	11.69%	4-0	4-2-0	4-2-1-0	4-2-3-1-0
4-2	90.75%	9.14%	0.10%	0.00%	4-0-2	4-0-1-2	4-0-1-3-2		33.05%	26.88%	20.86%	19.20%	4-0-2	4-0-1-2	4-0-1-3-2	4-2
4-3	90.59%	9.29%	0.13%		4-0-1-3	4-0-6-1-3	4-2-3		38.91%	32.68%	28.41%		4-0-1-3	4-0-6-1-3	4-2-3	
5-0	100.00%	0.00%	0.00%	0.00%	5-0				100.00%	0.00%	0.00%	0.00%	5-0			
5-7	99.90%	0.10%	0.00%		5-0-7	5-6-0-7			97.60%	2.39%	0.00%		5-0-7	5-6-0-7		
6-0	100.00%	0.00%	0.00%	0.00%	6-0				100.00%	0.00%	0.00%	0.00%	6-0			
6-1	98.79%	0.66%	0.00%	0.55%	6-0-1	6-0-2-1		6-5-0-1	15.69%	0.16%	0.00%	84.15%	6-0-1	6-0-2-1		6-5-0-1
6-5	93.06%	6.89%	0.04%	0.00%	6-0-5	6-1-0-5			36.78%	22.02%	10.43%	30.76%	6-0-5	6-1-0-5	6-1-2-0-5	6-5
7-0	100.00%	0.00%			7-0				100.00%	0.00%			7-0			
7-8	88.42%	11.58%			7-0-8	7-8			48.99%	51.01%			7-0-8	7-8		
8-0	100.00%	0.00%			8-0				100.00%	0.00%			8-0			
8-6	64.91%	1.96%	33.13%	0.00%	8-0-1-6	8-0-5-6	8-0-6		1.72%	1.21%	96.48%	0.59%	8-0-1-6	8-0-5-6	8-0-6	8-7-0-6
8-7	16.58%	83.42%			8-0-7	8-7			2.06%	97.94%			8-0-7	8-7		

Chapter 5

Conclusions and Future Work

5.1 Summary and Conclusions

A two-phase approach combining the Linear Programming (LP) and the Hopfield Neural Network (HNN) technology has been designed to perform the optimization of the nation-wide multi-service Internet Backbone by maximizing the total network profit as well as the service performance. Firstly, a LP model has been formulated and solved, whose main aim is to find the maximum of the network profits. In this model, the voice service class is given a higher priority to use the entire bandwidths in order to satisfy its real-time requirements, while the data service class has to use the residual bandwidth. In addition, the characteristics of the voice over IP service are taken into account specially by setting the bandwidth carriers on both directions between a pair of cities to be the larger one. Besides the optimal profit, the traffic loaded between each pair of cities and the actual link capacity assigned are also obtained. Next, based on the results of the LP optimization, an HNN model is addressed to take care of the further optimization of the performance of the data service class in order to increase the overall performance quality. In order to simplify the expression of the total throughput, a summation of the data traffic staying in the network system is used as an indication and explicitly included into the final energy function. With the linear HNN model, all the data packets are routed through different routing paths so that the throughput of the data service is also maximized when the total network profit is the highest. Since network survivability is becoming more and more important, an algorithm based on the repeated applying Dijkstra's algorithm is also

proposed for routing selection when the HNN model is designed and implemented. Trying to spread the traffic into those routes passing different links, this algorithm can provide a better performance in the face of network failure. Finally, a friendly user interface helps users to do the optimization and comparison experiments in an easier manner.

The proposed method has been applied to the Colombian Internet Backbone system, and a large number of experiments have been performed. The results are highly encouraging and highlight our combination approach is a promising and high effective tool for current nation-wide Internet Backbone systems.

In conclusion, we have presented a combined cost-effective design with LP and HNN models for maximizing the overall Internet Backbone network profit that performs very well in considerations of service performance requirements compared with other schemes.

5.2 Future Work

The proposed approach is to find an optimal designed Internet Backbone system supplying multiple services and also well-handling QoS requirements in loading traffic. The investigation issues for the future work could be as follows.

In our thesis, we only take two service classes into account between characterized by different QoS requirements. Nevertheless, there is great interest in evaluating the optimization for the current or the future Internet systems which integrates many other service classes or protocols. In addition, we have provided a bound for the block probability when we generated the total profit objective function in order to simplify the LP model. However, the exact impact of this parameter is unknown and should be included into a constraint with traffic flow and capacity.

We have shown that the extended optimization for the data packet routing so that the lack of consideration in the service quality of this class can be complemented. It would be more useful to have an HNN technology implementation in both of the service classes. Yet another topic could be the future work is the comparative performance with the dynamic traffic demand instead of the static data.

Bibliography

[1] *Michaela Plante, Brunilde Sansò*, "A Typology for Multi-technology, Multi-service Broadband Network Synthesis", *Telecommunication System*, pp1-37, 0(2000).

[2] *P. Jocher, Ch. Winkler*, "Topological Optimization of an ATM-based Connectionless Overlay Network", Proceedings 5th Open Workshop on High Speed Networks, Paris, France, pp. 1.25-1.33, March 1996

[3] *D. Mitra, K. G. Ramakrishman*, "Techniques for Traffic Engineering of Multiservice, Multipriority networks", *Bell Labs Technical Journal*, Jan-Jun 2001.

[4] *Mainak Chatterjee, Sajal K. Das, Giridhar D. Mandyam*, "Performance Evaluation of Voice-Data Integration for Wireless Data Networking", *First International Conference on Network (ICN), INCS 2093*, July 2001, pp157-166.

[5] *D. Mitra, John A. Morrison, K. G. Ramakrishnan*, "VPN Designer: A Tool for Design of Multiservice Virtual Private Networks", *Bell Labs Technical Journal*, pp15-31, Oct-Dec 1998.

[6] *Barberis, C.Casetti, J. C. De Martin, M.Mao*, "A Simulation Study of Adaptive Voice Communications on IP Networks", *Computer Communication*, pp757-767, 24(2001).

[7] *A. Bley, M. Grötschel, R. Wessäly*, "Design of Broadband Virtual Private Networks: Model and Heuristics for the B-WiN", *Preprint SC 98-13*, Mar 1998.

[8] *E. Thibault*, "A Decision Support System for the Design of Cost-Effective Metropolitan Area Networks", *Master thesis*, University of Ottawa, May 1999.

[9] *D.Alevras, M. Grötschel, R. Wessäly*, "A Network Dimensioning Tool", *Preprint SC 96-49, Konrad-Zuse-Zentrum für Information Stechrick Berlin*, Jan27, 1997.

[10] *Daniel Bienstock, Sunil Chopra, Oktay Gűnlűk, Chih-Yang Tsai*, "Minimum Cost Capacity Installation for Multicommodity Network Flows", *CORE Discussion Paper*, Université Catholique de Louvain, Belgium, 1995.

[11] *G. Nemhauser, L. A. Wolsey*, "Interger and Combinatorial Optimization", *John Wiley and Sons*, Newyork, 1988.

[12] *M-J Lee, J. R. Yee*, "An Efficient Near-Optimal Algorithm for the Joint Traffic and Trunk Routing Problem in Self-Planning Networks", Proc. *IEEE* INFOCOM'89, Ottawa, pp127-135, 1989.

[13] *Tony Lisotta, Mark Radwin*, "EOSDIS Technology Assessment Study WAN Network Technology", *NASA Ame Research Center*, 3/30/98.

[14] *Kate A. Smith*, "Neural Networks for Combinatorial Optimization: A Review of More Than a Decade of Research", *INFORMS Journal on Computing*, Vol.11, No.1, Winter 1999.

[15] *G. Feng, C. Douligers*, "A Neural Network Method for Minimum Delay Routing in Pack-Switched Networks", *Computer Communication*, pp933-941, 24(2001).

[16] *Jzau_Sheng Lin, Mingshou Liu, and Nen-Fu Huang*, "The Shortest-Path Computation in MOSPF Protocol through an Annealed Chaotic Neural Networks", *Proc. Natl. Sci. Counc. ROC(A)*, Vol. 24, No.6, pp.463-471, 2000.

[17] *Jeffrey E. Wieselthier, Anthony Ephremides*, "A Neural Network Approach to Routing without Interference in Multihop Radio Networks", *IEEE Transactions on Communications*, pp.166-171, 1994.

[18] *Jian-Kang Wu*, "Neural Network and Simulation Methods", *Marcel Dekker*, Inc. 1994.

[19] *Yu Liu, David Tipper, Peerapon Siripongwutikorn*, "Approximating Optimal Spare Capacity Allocation by Successive Survivable Routing", *IEEE Infocom*, April, 2001.

[20] *Ch. P. Sarantionopoulos, D. K. Karagiannis, K. Peppas, P. P. Demestichas, V. P. Demesticha, M. Theologou*, "Management and Control of the Interconnecting Network of the Access Segment of Future Mobile Communications Systems", *10-th Mediternanean Electrotechnical Conference, IEEE, MELECON 2000*, May 29-31, 2000 Cyprus.

[21] *Shigang Chen*, "Routing support for Providing Guaranteed End-to-End Quality-of-Service", *Ph.D. Thesis Report*, 1999.

[22] *D.Alevras, M. Grötschel, R. Wessäly*, "Capacity and Survivability Models for Telecommunication Networks", *Pre-print*, 1997.

[23] *Kevin Thompson, J. Miller, and Rick Wilder,* "Wide-Area Internet Traffic Patterns and Characteristics (Extended Version)", *IEEE Network,* November/December 1997.

[24] *Sally Floyd and Vern Paxson,* "Difficulties in Simulating the Internet", *IEEE/ACM Transactions on Networking,* February 2001.

[25] *Donald W. Hean,* "A Toll Pricing Framework for Traffic Assignment Problems with Elastic Demand", *Current Trends in Transportation and Network Analysis,* edited volume, January 3, 2002.

[26] *R. Elbaum, M. Sidi,* "Topological Design of Local Area Networks Using Generic Algorithms", *IEEE Proceedings*, pp.64-71, 1995.

[27] "Internet Telephony On-line Tutorial", http://www.iec.org/online/tutorials/int_tele.

[28] CPLEX callable library, http://www.ilog.com

[29] "OSPF Design Guide", http://www.cisco.com/wap/public/104/1.html.

[30] "Mixed Packet Size Throughput, Throughput and Latency", http://advanced .comms.agilent.com/routertester/member/journal/JTC_003.html.

[31] "Distinctive Features -- Dimensions On Which Connectionist Models of Psychopathology May Vary", http://www.cnbc.cmu.edu/disordermodels/distinctive-features.html.

[32] *Geng Feng, Christos Douligeris*, "Using Hopfield Networks to Solver Traveling Salesmam Problems Based on Stable State Analysis Technique", *Proc. Int. Joint Conf. Neural networks*, vol.6, pp. 521-526, July 2000.

[33] *J.J. Hopfield, D.W. Tank*, "Neural Computations of Decisions in Optimization Problems", *Biological Cyberbetics*, 52, pp.141-152, 1985.

[34] "Artificial Neural Network Technology", http://www.dacs.dtic.mil/techs/neural/neural2.html.

[35] "Hopfield's Networks", http://www.cz3.nus.edu.sg/~wangjs/CZ3205/Notes7_1.htm.

[36] OSPF Tutorial, http://www.geocities.com/Heartland/4394/work/ospf.html.

[37] Colombia Internet Backbone, http://www.punto-com.com.

Appendices

Appendix A: CPLEX

1. CPLEX

CPLEX is a solver of the linear optimization problems of the form:

maximize or minimize	$\sum_i c_i x_i$
Subject to	$\sum_i a_{1i} x_i \sim b_1$
	$\sum_i a_{2i} x_i \sim b_2$
	...
	$\sum_i a_{mi} x_i \sim b_m$
Bounds	$l_1 \le x_1 \le m_1$
	$l_n \le x_n \le m_n$

where $\sim$ can be $\le, \ge,$ *or* $=$, and the upper bound u_i and lower bound l_i may be positive or negative infinity, or any real number. The following are the constants of this linear program:

Objective function coefficients	$c_i \quad (i \in 1, 2, ..., n)$
Constraint coefficients	$a_{1i}, \cdots, a_{mi}$
Right-hand side	b_i
Upper and lower bounds	u_i, *and* l_i

The unknowns are:

Variables x_i

The CPLEX Base System algorithms and Barrier Solver solve problems with continuous variables (not restricted to integer values), whereas the Mixed Integer Solver allows both continuous and integer variables. Variables can be two types: binary for those restricted to values 0 and 1, and general integer for others. The CPLEX Barrier Solver also solves problems with quadratic objective terms.

2. CPLEX Callable Library

The CPLEX Callable Library is specifically designed to facilitate the development of applications to solve, modify, and interpret the results of linear, mixed integer and convex quadratic programming programs. The CPLEX Callable Library together with the CPLEX database make to the CPLEX Core. The Core becomes associated with user's application through the proper use of the Library routines. The CPLEX environment and all problem defining data are established inside of the CPLEX Core.

The CPLEX Callable Library comprises six categories of routines:

1. Optimization and result routines for defining a problem, optimizing it, and getting the results;
2. Utility routines for handling I/O and addressing application programming matters;
3. Problem modification routines to change a problem once it has been created within the CPLEX database;
4. Problem query routines to access information about a problem once it has been created;
5. File reading and writing routines to move information from the file system into your application, or out of your application to the file system; and,
6. Parameter setting and query routines to access and modify the values of control parameters maintained by CPLEX.

3. Routines Used in the Thesis

Creating Problems-

CPXcreateprob create a problem object in the CPLEX environment. The problem created is an LP minimization problem with zero constraints, zero variables, and an empty constraint matrix.

Accessing LP results-

CPXsolution access optimal LP solution values, if a solution exists, returns 0; if no solution exists or some other failure occurs, returns a nonzero value.

CPXgetx access optimal variable values, the beginning and end of the range must be specified. The routine returns a 0 on success, and a nonzero if an error occurs.

Message Handling Routine-

CPXmsg the only routine not requiring the CPLEX environment parameter, write a message to a specified channel, returns the number of characters in the formatted result string.

Problem Modification Routines-

CPXnewrows add empty constraints to a specified problem object without matrix coefficients, return 0 on success, and a nonzero if an error occurs.

CPXaddrows add constraints to a specified problem object, returns a 0 on success, and a nonzero if an error occurs.

CPXnewcols add empty columns to a specified problem object; can be used to add variables without specifying the matrix coefficients. For each column, the user can specify the objective coefficient, the lower and upper bounds, the variable type, and the name of the variables.

CPXaddcols add variables to a specified problem object, returns a 0 on success, and a nonzero if an error occurs.

Appendix B: OSPF

This appendix is extracted from the materials in Ref [36].

Open Shortest Path First (OSPF) is a routing protocol to determine the correct route for packets within IP networks. It was designed by the Internet Engineering Task Force to serve as an Interior Gateway Protocol replacing RIP.

Advantages:

1. Propagate changes in a network very quickly.
2. OSPF is hierarchical, where 0 is the top of the hierarchy.
3. OSPF is a Link State Algorithm.
4. OSPF supports Variable Length Subnet Masks (VLSM).
5. OSPF uses multicasting within areas.
6. After initialization, OSPF only sends update on routing table sections which have changed, not the entire routing table.
7. Using areas, OSPF networks can be logically segmented to decrease the size of routing tables. Table size can be further reduced by using route summarization.
8. OSPF is an open standard, not related to any particular vendor.

Disadvantages:

1. OSPF is very processor intensive.
2. OSPF maintains multiple copies of routing information, increasing the amount of memory needed.
3. OSPF is not easy to learn.
4. When one link within an OSPF network is "bouncing" every few seconds, OSPF updates would inform each other router every time the link changed state.

OSPF routers check the status of other routers on the network by sending a small hello packet at regular intervals. If a router does not respond, it is assumed dead, and routing updates are sent to every other router by using a multicast address.

If there are no network changes, OSPF only uses very little bandwidth (only sending hello packet). As soon as there is an outage, however, OSPF will flood the network since the change is sent to every router (and then every router notifies every other router about the change). This system of near silence when possible and flooding when necessary ensures that routing information gets propagated throughout the network as quickly as possible.

Appendix C: Dijkstra's Shortest-Path Algorithm

A Dutch computer scientist and mathematician, Edsger W. Dijkstra proposed a strategy for getting the shortest path from node s to node r by finding the shortest path from s to all other nodes in the graph.

The main idea of this algorithm is to change the temporary labels associated with nodes into permanent ones. The permanent label of a node denotes the shortest path weight from the source node to the current node. For node i, we denote:

$$A(i) = \{j : e = (i, j) \in E, j \text{ has the temporary label}\}.$$

Initially, node r has a permanent label 0, while $j \in A(r)$ with a temporary label $t(r, j)$, and all other nodes with a temporary label ∞. Let P to be the set of all the nodes with permanent labels, and $T = V - P$ to be the set of all the nodes with temporary labels. At each step, the algorithm chooses the node $i \in T$ with the minimum temporary label, and makes it permanent, record its predecessor index, and update the temporary values of all the node $j \in A(i)$. Repeat this procedure until all nodes become permanent ones.

Algorithm Dijkstra {
  $P = \{s\}$; $T = V - \{s\}$;
  $d(s) = 0$; $p(r) = 0$;
  $d(j) = c_{rj}$;
  $p(j) = s$ *for all* $(r, j) \in A$, *and* $d(j) = \infty$ *for other nodes*;
  while ($P \neq V$) *do*
  {
    choose the minimum $i \in T$
    $d(i) = min\ \{d(j) : j \in T\}$
    $P = P \cup \{i\}, T = T - \{i\}$
    for all $j \in A(i)$, *compute* $d(j) = min\ \{d(j), d(i) + c_{ij}\}$,
    set $p(j) = i$;
  }
}

The example source code is easily obtained from the Internet.

Appendix D: Numerical Code

1. coltest.c (for voice service)

```
#include <stdio.h>
#include <stdlib.h>

#include <ilcplex/cplex.h>

#define alfa        0.25
#define delay       60          //maximum acceptable delay: 60ms
#define cu          155.0       //average capacity unit size: 155.0kbps
#define ps          400         //average packet size: 400bytes

#ifndef CPX_PROTOTYPE_MIN
int
gennt (int sd, int ij, int *rout, double *traffic, double *capacity, double *deltv,
       int *flatbeta, CPXENVptr env, CPXLPptr lp)

#else
int
gennt (sd, ij, rout, traffic, capacity, deltv, flatbeta, env, lp)
int         sd;
int         ij;
int         *rout;
double      *traffic;
double      *capacity;
double      *deltv;
int         *flatbeta;

CPXENVptr   env;
CPXLPptr    lp;

#endif
{
    double *obj     = NULL;
    char   *sense   = NULL;
    double *rhs     = NULL;
    int    *cmatbeg = NULL;
    int    *cmatind = NULL;
    double *cmatval = NULL;
    double *lb      = NULL;
    double *ub      = NULL;

    CPXCHANNELptr   cpxerror = NULL;

    int s, i, r, m;         /* loop counters */
    int t;                  /* counter for flatbeta index */
    int numcolsV, numnzV;   /* for Vof memory sizing */

    int status = 0;
    status = CPXgetchannels (env, NULL, NULL, &cpxerror, NULL);
    if (status) goto TERMINATE;

    /* Set the objective sense to be minimize */
    CPXchgobjsen (env, lp, CPX_MIN);

    /* Set up the constraints for the problem */
    /* First do the capacity constraints. Allocate space for a sense array */
    sense = (char   *) malloc ((ij+sd)*sizeof (char));
    rhs   = (double *) malloc ((ij+sd)*sizeof (double));

    if (   sense == NULL ||
           rhs   == NULL   ) {
           status = -2;
           goto TERMINATE;
```

```
}

for ( i = 0; i < ij; i++ ) {
        sense[i] = 'L';
        rhs[i]   = capacity[i]-ps*8/delay;
}

for ( s = ij; s < sd+ij; s++ ) {
        sense[s] = 'L';
        rhs[s]   = traffic[s-ij];
}

status = CPXnewrows (env, lp, ij+sd, rhs, sense, NULL, NULL);
if ( status ) {
        CPXmsg (cpxerror, "CPXnewrows failed to add other traffic constraints\n",
                status);
        goto TERMINATE;
}

/* Free up the temporary sense array */
free (sense); sense = NULL;

/* Now add the variables. For each set of variables has objective, we allocate
 * arrays to hold the data for the CPXaddcols() calls, and then free them up, so
 * that each set of variables is maintained in a "local" piece of code.
 */

 /* Do the Vof variables. */
numcolsV = 0;
numnzV   = 0;
for (s = 0; s < sd; s++) {
         numcolsV += rout[s];
}

for (s = 0; s < numcolsV; s++) {
        for (i = 0; i < ij; i++) {
                if (flatbeta[i*numcolsV+i] == 1) {
                        numnzV +=1;
                }
        }
}

obj     =  (double *) malloc ( numcolsV * sizeof(double));
cmatbeg = (int    *) malloc ( numcolsV * sizeof(int));
cmatind = (int    *) malloc ( numnzV   * sizeof(int));
cmatval = (double *) malloc ( numnzV   * sizeof(double));

if ( obj     == NULL ||
     cmatbeg == NULL ||
     cmatind == NULL ||
     cmatval == NULL   ) {
    status = -2;
        goto TERMINATE;
}

t = 0;          /* counter of flatbeta index */
m = 0;
for (s = 0; s < sd; s++) {
        for (r = 0; r < rout[s]; r++) {
            cmatbeg[t] = m;
            obj[t]     = -1.0*deltv[s];
           for (i = 0; i < ij; i++) {
               if (flatbeta[t*ij+i]==1) {
                   cmatind[m] = i;
                   cmatval[m] = 1.0*flatbeta[t*ij+i];
                   m++;
               }
            }
            cmatind[m] = ij+s;
            cmatval[m] = 1.0/alfa;
            m++;
```

```
                t+=1;
            }
        }

        // CPXaddcols(env, lp, number of beg, number of val, obj,beg, ind, val, lb, ub,
        // colname)
          status = CPXaddcols (env, lp, numcolsV, numnzV, obj, cmatbeg, cmatind, cmatval,
                               NULL, NULL, NULL);
          if (status) {
               CPXmsg (cpxerror, "CPXaddcols, failed to add Vof variables\n",status);
               goto TERMINATE;
          }

       /* Free up allocated memory used to add Vof variables */
         free (obj);     obj     = NULL;
         free (cmatbeg); cmatbeg = NULL;
         free (cmatind); cmatind = NULL;
         free (cmatval); cmatval = NULL;

TERMINATE:

        if (obj     != NULL)  free (obj);
        if (sense   != NULL)  free (sense);
        if (cmatbeg != NULL)  free (cmatbeg);
        if (cmatind != NULL)  free (cmatind);
        if (cmatval != NULL)  free (cmatval);
        if (lb      != NULL)  free (lb);
        if (ub      != NULL)  free (ub);

        return (status);
} /* End GenNt */
```

2. test1drv.c (for voice service)

```
#include <stdio.h>
#include <stdlib.h>

/* The following is needed to get the CPLEX constants */
#include <ilcplex/cplex.h>

/* Declare external function */
#ifndef CPX_PROTOTYPE_MIN
int
    nt_opt_and_soln ( void (CPXPUBLIC *errmsgfunc)(void *, char *), void *handle,
                      int sd, int ij, int *rout,double *traffic, double *capacity,
                      double *deltv, int *flatbeta,
                      int *lpstat_p, double *objval_p, double *Vof );

static void CPXPUBLIC
    errmsgfunc (void *handle, char *string);

#else
int
    nt_opt_and_soln ();

static void CPXPUBLIC
    errmsgfunc ();
#endif

#define MAXROUT         7
//#define NODE            9
#define SD              31
#define IJ              28
#define NUMCOLSV        146
#define cu              155.0

#ifndef CPX_PROTOTYPE_MIN
int
main()
```

```
{
#endif

/* InputData */
int    rout[SD];
double traffic[SD];
double capacity[IJ];
double deltv[SD];
int    flatbeta[NUMCOLSV*IJ];

#ifdef CPX_PROTOTYPE_MIN
int
main()
{
#endif

   double numUnit[IJ];
   double vof[NUMCOLSV];

   int    lpstat;
   double objval;
   int    s, i, r, t;
    /* various loop counters, s:SD, i:IJ, r:ROUT */

   int    status;

   //double cost;      /* parts of the objective */
   double revenueV=0;

   for (s = 0; s < NUMCOLSV; s++) {
      vof[s]=0.0;
   }

   status = nt_opt_and_soln (errmsgfunc, stderr, SD, IJ, rout, traffic, capacity,
                             deltv, flatbeta, &lpstat, &objval, &vof[0]);

   if (status) goto TERMINATE;

   printf ("Solution status: %d\n", lpstat);

   if (lpstat != CPX_OPTIMAL) {
      printf ("Solution not optimal!!\n");
      goto TERMINATE;
   }

   printf ("objective value: %g\n", objval);
   printf ("\n");

   printf ("\nFlow of Voice: \n%5s ", "Sour-Dest");
   for (r = 0; r < 7; r++) {
       printf ("        ROUT%2d    ", r);
   }
   printf ("\n");

   t=0;
   for (s = 0; s < SD; s++) {
       printf ("%3d ", s);
       for (r = 0; r < rout[s]; r++) {
              revenueV += 1.0*vof[t+r];
              printf ("    %12g   ", vof[t+r]);
       }
       t+=rout[s];
       printf ("\n");
       printf("------------------------------------------------------------------------
------\n");
   }
   printf ("objective of voice revenue:%g\n", revenueV);
   printf ("\n");

   /* calculate the capacity unit number for each link */
   t = 0;
```

```
   for (i = 0; i < IJ; i++) {
       for (s = 0; s < SD; s++) {
              for (r = 0; r < rout[s]; r++) {
                     numUnit[i] += vof[t+r]*flatbeta[t*IJ+r*IJ+i]/cu;
              }
       }
   }
   printf ("LINK      Number of Unit \n");
   for (i = 0; i < IJ; i++ ) {
       printf ("%2d          %5g", i, numUnit[i]);
       printf ("\n");
   }

TERMINATE:
   return (status);
} /* End Main */

/* This is a very simple message handler to just print the
 * message to the file pointed to by handle.
 */

#ifndef CPX_PROTOTYPE_MIN
static void CPXPUBLIC
errmsgfunc (void *handle, char *string)
#else
static void CPXPUBLIC
errmsgfunc (handle, string);
void *handle;
char *string;
#endif
{
     FILE *fp;

     fp = (FILE *) handle;
  /* Convert handle to a FILE * type */
     fprintf (fp, "%s", string);
} /* End errmsgfunc */
```

3. testsub.c (for voice service)

```
#include <ilcplex/cplex.h>
#include <stdlib.h>

#ifndef  CPX_PROTOTYP_MIN
int
   gennt (int sd, int ij, int *rout, double *traffic, double *capacity,
          double *deltv, int *flatbeta, CPXENVptr env, CPXLPptr lp);
#else

int
   gennt();
#endif

#ifndef CPX_PROTOTYP_MIN
int
nt_opt_and_soln (void (CPXPUBLIC *errmsgfunc)(void *, char*),void *handle, int sd,
                 int ij, int *rout, double *traffic, double *capacity, double *deltv,
                 int *flatbeta, int *lpstat_p, double *objval_p, double *Vof)

#else
int
nt_opt_and_soln (errmsgfunc, handle, sd, ij, rout, traffic, capacity, deltv, flatbeta,
                 lpstat_p, objval_p, Vof)

void    (CPXPUBLIC *errmsgfunc)();
void    *handle;
int     sd;
int     ij;
int     *rout;
double  *traffic;
```

```
double  *capacity;
double  *deltv;
int     *flatbeta;

int     *lpstat_p;
double  *objval_p;
double  *Vof;
#endif

{
     char  *probname = "NetworkOp";

     CPXENVptr       env =         NULL;
     CPXLPptr        lp  =         NULL;
     CPXCHANNELptr   cpxerror   = NULL;
     CPXCHANNELptr   cpxwarning = NULL;

     int     status = 0;
     int     lpstat;
     double  objval;

     int     numcolsV, s;  /* for Vof memory sizing, s is a loop counter */

     env = CPXopenCPLEX (&status);

     /* If an error occurs, the status value indicates the reason for failure. We'll
      * call the errmsgfunc with the erro, because we can't call CPXmsg if
      * CPXopenCPLEXdevelop failed.
      */
     if (env == NULL) {
            char errmsg[1024];
            if (errmsgfunc != NULL) {
                   (*errmsgfunc) (handle, "Could not open CPLEX environment.\n");
                   CPXgeterrorstring (env, status, errmsg);
                   (*errmsgfunc) (handle, errmsg);
            }
           goto TERMINATE;
     }

     status = CPXgetchannels (env, NULL, &cpxwarning, &cpxerror, NULL);
     if (status) {
            if (errmsgfunc != NULL) {
                   (*errmsgfunc) (handle, "Error in CPXgetchannels.\n");
            }
            goto TERMINATE;
     }

     if (errmsgfunc != NULL) {
            status = CPXaddfuncdest (env, cpxerror, handle, errmsgfunc);
            if (!status) {
                   status = CPXaddfuncdest (env, cpxwarning, handle, errmsgfunc);
            }

        if (status) {
              (*errmsgfunc) (handle, "Error in CPXaddfuncdest.\n");
               goto TERMINATE;
        }
      }

     /* Now the channels have the error message function, we can use CPXmsg for
errors.
      * Create the problem
      */

      lp = CPXcreateprob (env, &status, probname);

      if (lp == NULL) {
            status = -2;
            CPXmsg (cpxerror, "Failed to create LP.\n");
            goto TERMINATE;
      }
```

```
      status = gennt (sd, ij, rout, traffic, capacity, deltv, flatbeta, env, lp);

      if (status) goto TERMINATE;

      status = CPXlpopt (env, lp);
      if (status) {
            CPXmsg (cpxerror, "Optimization failed. Status %d.\n", status);
            goto TERMINATE;
      }

      //CPXsolution (env, lp, &lpstat, &objval, NULL, NULL, NULL, NULL)
      status = CPXsolution (env, lp, &lpstat, &objval, NULL, NULL, NULL, NULL);
      if (status) {
            CPXmsg (cpxerror, "Solution failed. Status %d.\n", status);
            goto TERMINATE;
      }

      *objval_p = objval;
      *lpstat_p = lpstat;

     /* Now get the solution. Use the fact that the variables are added in the order
      * NumUnit, Vof, Daf, and the sloces of the X vector can be used to copy over the
      * solution into the flat...arrays.
      */

     /* Get the Vof variables */
     numcolsV = 146;
     status = CPXgetx (env, lp, Vof, 0, numcolsV-1);

     if (status) {
         CPXmsg (cpxerror, "CPXgetx failed to get Vof solution. status %d.\n",
status);
         goto TERMINATE;
     }

TERMINATE:

      if (lp != NULL)  CPXfreeprob (env, &lp);

      if (errmsgfunc != NULL && env != NULL) {
            int dfstat = CPXdelfuncdest (env, cpxerror, handle, errmsgfunc);
            if (!dfstat) {
                  dfstat = CPXdelfuncdest (env, cpxwarning, handle, errmsgfunc);
            }
            if (dfstat) {
                  (*errmsgfunc) (handle, "CPXdelfuncdest failed.\n");
            }
            if (dfstat && !status)  status = dfstat;
      }

     /* free up the CPLEX environment, if necessary */
     if (env != NULL) {
           int closestat = CPXcloseCPLEX (&env);

           if (closestat && errmsgfunc != NULL) {
                 char errmsg[1024];
                 (*errmsgfunc) (handle, "Could not close CPLEX environment.\n");
                 CPXgeterrorstring (env, closestat, errmsg);
                 (*errmsgfunc) (handle, errmsg);
           }
           if (closestat && !status) status = closestat;
     }

     return (status);
}  /* End nt_opt_and_soln */
```

4. coltestd.c

```
#include <stdio.h>
#include <stdlib.h>
#include <ilcplex/cplex.h>

#define alfa        0.25
#define delay       60      //maximum acceptable delay: 60ms
#define cu          155.0   //average capacity unit size: 155.0kbps
#define ps          400     //average packet size: 400bytes

#ifndef CPX_PROTOTYPE_MIN
int
gennt (int sd, int ij, int node, int *rout, double *traffic, double *capacity,
       double *deltd, double k, double *NumUnit, int *indi, int *io, int *Into,
       int *Outof, int *ns, int *nd, int *Sour, int *Dest, CPXENVptr env, CPXLPptr lp)

#else
int
gennt (sd, ij, node, rout, traffic, capacity, deltv, deltd, k, NumUnit, indi, io,
Into,
       Outof, ns, nd, Sour, Dest, env, lp)

int         sd;
int         ij;
int         node;
int         *rout;
double      *traffic;
double      *capacity;
double      *deltd;
double      k;
double      *NumUnit;
int         *indi;
int         *io;
int         *Into;
int         *Outof;
int         *ns;
int         *nd;
int         *Sour;
int         *Dest;

CPXENVptr   env;
CPXLPptr    lp;
#endif
{
    double *obj     = NULL;
    char   *sense   = NULL;
    double *rhs     = NULL;
    int    *cmatbeg = NULL;
    int    *cmatind = NULL;
    double *cmatval = NULL;
    double *lb      = NULL;
    double *ub      = NULL;

    CPXCHANNELptr   cpxerror = NULL;

    int s, i, n, r, m;     /* loop counters */
    int t;                 /* counter for flatbeta index */
    double *recap;
    int numcolsD, numnzD;  /* for Daf memory sizing */
    int Lio(int i, int n); /* function for indicating if the ith link into or */
                           /* outof the nth node */
    int Nsd(int n, int s); /* function for indicating if the nth node */
                           /* is the destination or source of the sth nodepair */

    int status = 0;

    status = CPXgetchannels (env, NULL, NULL, &cpxerror, NULL);
    if (status) goto TERMINATE;

    /* Set the objective sense to be minimize */
    CPXchgobjsen (env, lp, CPX_MIN);
```

```
/* Set up the constraints for the problem */
/* First do the capacity constraints. Allocate space for a sense array */

sense = (char   *) malloc ((ij+sd*node*3)*sizeof (char));
rhs   = (double *) malloc ((ij+sd*node*3)*sizeof (double));

if ( sense == NULL || rhs   == NULL   ) {
       status = -2;
       goto TERMINATE;
}

for (i = 0; i < ij; i++) {
       recap[i] = capacity[i]-NumUnit[i];
}

for ( i = 0; i < ij; i++ ) {
       sense[i] = 'G';
       rhs[i]   = ps/delay;
}

i = 0;
for (n = 0; n < node; n++) {
       for (s = 0; s < sd; s++) {
            sense[ij+i] = 'L';
            rhs[ij+i]   = traffic[s];
            i++;
            sense[ij+i] = 'G';
            rhs[ij+i]   = -traffic[s];
            i++;
            sense[ij+i] = 'E';
            rhs[ij+i]   = 0.0;
            i++;
       }
}

status = CPXnewrows (env, lp, ij+sd*3*node, rhs, sense, NULL, NULL);
if ( status ) {
       CPXmsg (cpxerror, CPXnewrows failed to add other traffic constraints\n",
               status);
       goto TERMINATE;
}

/* Free up the temporary sense array */
free (sense); sense = NULL;

/* Now add the variables. For each set of variables has objective, we allocate
 * arrays to hold the data for the CPXaddcols() calls, and then free them up, so
 * that each set of variables is maintained in a "local" piece of code.
 */

/* First, do the NumUnit variables. Each NumUnit[i] variable has objective
 * coefficient k, and bounds of (0, capacity[i]/cu). Each NumUnit[i] variable
 * appears in the constraints capacity[i].
 * Create temporary arrays to hold objective coefficients, lower and upper bounds,
 * and matrix coefficients for one NumUnit.
 */
obj     = (double *) malloc (ij * sizeof (double));
cmatbeg = (int    *) malloc (ij * sizeof (int));
cmatind = (int    *) malloc (ij * sizeof (int));
cmatval = (double *) malloc (ij * sizeof (double));

if ( obj     == NULL ||
     cmatbeg == NULL ||
     cmatind == NULL ||
     cmatval == NULL    ) {
       status = -2;
       goto TERMINATE;
}

m = 0;
for (i = 0; i < ij; i++ ) {
```

```
        obj[i]      = k;
        cmatbeg[i] = m;
        cmatind[m] = i;
        cmatval[m] = cu;
        m++;
  }
  //  CPXaddcols(env, lp, number of beg, number of  val,obj, beg, ind, val, lb, ub,
  //colname)
     status = CPXaddcols (env, lp, ij, m, obj, cmatbeg, cmatind, cmatval, NULL,
                          recap, NULL);

     if (status) {
         CPXmsg (cpxerror, "CPXaddcols, failed to add NumUnit variables\n",
                  status);
         goto TERMINATE;
     }

  /* Free up allocated memory used to add NumUnit variables */
  free (obj);      obj      = NULL;
  free (cmatbeg); cmatbeg = NULL;
  free (cmatind); cmatind = NULL;
  free (cmatval); cmatval = NULL;
  free (ub);       ub       = NULL;

/* Do the Daf variables  */
 numcolsD = sd * ij;
 numnzD   = sd * ij * (3*node+1);
 obj      = (double *) malloc ( numcolsD * sizeof (double));
 cmatbeg = (int    *) malloc ( numcolsD * sizeof (int   ));
 cmatind = (int    *) malloc ( numnzD   * sizeof (int   ));
 cmatval = (double *) malloc ( numnzD   * sizeof (double));

 if (obj     == NULL ||
     cmatbeg == NULL ||
     cmatind == NULL ||
     cmatval == NULL   ) {
            status = -2;
            goto TERMINATE;
 }

 m = 0;
 for (s = 0; s < sd; s++) {
        n = -1;
        do {
        n +=1;
        }
        while (Nsd(n, s) == 1 || n == node-1)
        t = n;

        for (i = 0; i < ij; i++) {
         cmatbeg[s*ij+i] = m;
         obj[s*ij+i]     = -1.0*deltd[s]*indi[s*ij+i]*Lio(i, t);
         cmatind[m]      = i;
         cmatval[m]      = -1.0*indi[s*ij+i];
         m++;
         for (n = 0; n < node; n++) {
              if (Nsd(n, s) == 1) {
                cmatind[m] = ij+s*node*3+n*3;
              //if (Lio(i,n)==1 || Lio(i,n)==-1) {
                  cmatval[m] = Lio(i,n)*indi[s*ij+i]/(ro*(1-alfa));
                  m++;
              //}
              }
              else if (Nsd(n, s) == -1) {
                 cmatind[m] = ij+s*node*3+n*3+1;
               //if (Lio(i,n)==1 || Lio(i,n)==-1) {
                  cmatval[m] = -Lio(i,n)*indi[s*ij+i]/(ro*(1-alfa));
                  m++;
               //}
              }
              else if (Nsd(n, s) == 2) {
```

```
                    cmatind[m] = ij+s*node*3+n*3+2;
                  //if (Lio(i,n)==1 || Lio(i,n)==-1) {
                    cmatval[m] = Lio(i,n)*indi[s*ij+i];
                    m++;
                  //}
                  }
                  else {
                     printf("error of Nsd\n");
                  }
              }
          }
      }
      printf("good\n");

// CPXaddcols(env, lp, number of beg, number of val, obj, beg, ind, val, lb, ub,
// colname)
      status = CPXaddcols (env, lp, numcolsD, numnzD, obj, cmatbeg,cmatind, cmatval,
                           NULL, NULL, NULL);
      if (status) {
             CPXmsg (cpxerror, "CPXaddcols, failed to add Daf variables\n", status);
             goto TERMINATE;
      }
      printf("good\n");

      /* Free up allocated memory used to add Daf variables */

      free (obj);     obj     = NULL;
      free (cmatbeg); cmatbeg = NULL;
      free (cmatind); cmatind = NULL;
      free (cmatval); cmatval = NULL;

TERMINATE:

      if (obj     != NULL)  free (obj);
      if (sense   != NULL)  free (sense);
      if (cmatbeg != NULL)  free (cmatbeg);
      if (cmatind != NULL)  free (cmatind);
      if (cmatval != NULL)  free (cmatval);
      if (lb      != NULL)  free (lb);
      if (ub      != NULL)  free (ub);

      return (status);
} /* End GenNt */

/* Lio function */
Lio(i,n) {
  int t, m, lio;

  int io[9];  // number of links into or outof each node
  int Into[28];       // links into each node
  int Outof[28];       // links outof each node

  m = 0;
  if (n != 0) {
    for (t = 0; t < n; t++) {
           m+=io[t];
    }
  }
  t = 0;
    while (t < io[n]) {
        if (i == Into[m+t]) {
               lio=1;
               goto TOP;
        }
        else if (i == Outof[m+t]) {
               lio=-1;
               goto TOP;
        }
        t +=1;
    }
    lio=0;
```

```
TOP:
        return (lio);
//      return 1;

    }

    /* Nsd function */
    Nsd(n,s) {
      int t, m1, m2, nsd;
      int Sour[31];    // each line means links with the nth node as source
      int Dest[31];    // each line means links with the nth node as destination
      int ns[9];       // number in each line of set Source[SD]
      int nd[9];       // number in each line of set Destination[SD]

      m1 = 0;
      m2 = 0;
      if (n != 0) {
        for (t = 0; t < n; t++) {
            m1+=ns[t];
            m2+=nd[t];
        }
      }

      t = 0;
      while (t < ns[n]) {
         if (s == Sour[m1+t]) {
            nsd=-1;
            goto BOTOM;
         }
         t+=1;
      }

      t = 0;
      while (t < nd[n]) {
         if (s == Dest[m2+t]) {
            nsd=1;
            goto BOTOM;
        }
        t+=1;
      }
      nsd=2;

    BOTOM:
        return (nsd);
        //return 1;
    }
```

5. testdrvd.c

```
#include <stdio.h>
#include <stdlib.h>

/* The following is needed to get the CPLEX constants */
#include <ilcplex/cplex.h>

/* Declare external function */
#ifndef CPX_PROTOTYPE_MIN
int
  nt_opt_and_soln ( void (CPXPUBLIC *errmsgfunc)(void *, char *), void *handle,
                    int sd, int ij, int node, int *rout, double *traffic,
                    double *capacity, double *deltd, double k, double *NumUnit,
                    double *Vof, int *flatbeta, int *indi, int *io, int *Into,
                    int *Outof, int *ns, int *nd, int *Sour, int *Dest,
                    int *lpstat_p, double *objval_p, double *NumUnitc, double *Daf);

static void CPXPUBLIC
    errmsgfunc (void *handle, char *string);

#else
```

```
int
    nt_opt_and_soln ();

static void CPXPUBLIC
    errmsgfunc ();
#endif

#define MAXROUT         7
#define NODE            9
#define SD              31
#define IJ              28
#define NUMCOLSV        146

#ifndef CPX_PROTOTYPE_MIN
int
main()
{
#endif

/* InputData */
int     rout[SD];
double traffic[SD;

/* need modified, -NumUnit*cu */
double capacity[IJ];
double deltd[SD;
double k;
int     indi[SD*IJ;
int     io[NODE];
int     Into[IJ];
int     Outof[IJ];
int     Sour[SD];
int     Dest[SD];
int     ns[NODE];
int     nd[NODE];

#ifdef CPX_PROTOTYPE_MIN
int
main()
{
#endif

   double numUnitc[IJ];
   double daf[SD*IJ];

   int     lpstat;
   double objval;

   int     s, i, r, t;       /* various loop counters, s:SD, i:IJ, r:ROUT */
   int     status;
   double ddaf[IJ];
   double cost;              /* three parts of the objective */
   double revenueD;

   for (s = 0; s < SD*IJ; s++) {
          daf[s]=0.0;
   }

   status = nt_opt_and_soln ( errmsgfunc, stderr, SD, IJ, NODE, rout, traffic,
                              capacity,deltd, k, NumUnit, indi, io, Into, Outof,
                              ns, nd, Sour, Dest, &lpstat, &objval, &numUnitc[0],
                              &daf[0]);

   if (status) goto TERMINATE;

   for (i = 0; i < IJ; i++) {
       for (s = 0; s < SD; s++) {
              ddaf[i] += daf[s*IJ+i];
       }
   }
```

```
   printf ("Solution status: %d\n", lpstat);
   if (lpstat != CPX_OPTIMAL) {
      printf ("Solution not optimal!!\n");
      goto TERMINATE;
   }

   printf ("objective value: %g\n", objval);
   printf ("\n");
   printf ("LINK      Number of Unit      Flow of Data\n");
   for (i = 0; i < IJ; i++ ) {
       cost += numUnitc[i] * k;
       printf ("%2d          %5g             %5g", i, numUnitc[i], ddaf[i]);
       printf ("\n");
   }

   printf ("objective of cost:%d\n", cost);
   printf ("\n");

   for (r = 0; r < 7; r++) {
       printf ("\nDistribution of data:\n%5s ", "Sour-Dest");
       for (i = 0; i < 4; i++) {
              printf ("        LINK%2d    ", i+r*4);
       }
       printf ("\n");
       for (s = 0; s < SD; s++) {
              printf ("%3d ", s);
              for (i = 0; i < 4; i++) {
                     revenueD += daf[s*4+i+SD*4*r] * indi[s*4+i+SD*4*r];
                     printf ("    %12g   ", daf[s*4+i+SD*4*r]);
              }
              printf ("\n");
              printf("-------------------------------------------------------------------
-------------\n");
       }

       printf ("\n");
   }
   printf ("revenue of data:%g\n", revenueD);
   printf ("\n");

   /*printf ("LINK      Flow of Data\n");
   for (i = 0; i < IJ; i++) {
       printf ("%2d          %5g ", i, ddaf[i]);
       printf ("\n");
   }
   printf ("\n");*/

TERMINATE:
   return (status);
} /* End Main */

/* This is a very simple message handler to just print the message
 * to the file pointed to by handle.
 */

#ifndef CPX_PROTOTYPE_MIN
static void CPXPUBLIC
errmsgfunc (void *handle, char *string)
#else
static void CPXPUBLIC
errmsgfunc (handle, string);
void *handle;
char *string;
#endif
{
     FILE *fp;

     fp = (FILE *) handle;      /* Convert handle to a FILE * type */
     fprintf (fp, "%s", string);
} /* End errmsgfunc */
```

6. testsubd.c

```
#include <ilcplex/cplex.h>
#include <stdlib.h>
#ifndef  CPX_PROTOTYP_MIN
int
   gennt (int sd, int ij, int node, int *rout, double *traffic, double *capacity,
          double *deltd, double k, double *NumUnit, double *Vof, int *flatbeta,
          int *indi, int *io, int *Into, int *Outof, int *ns, int *nd, int *Sour,
          int *Dest, CPXENVptr env, CPXLPptr lp);

#else
int
     gennt();
#endif

#ifndef CPX_PROTOTYP_MIN
int
nt_opt_and_soln (void (CPXPUBLIC *errmsgfunc)(void *, char *),void *handle,
                 int sd, int ij, int node, int *rout, double *traffic,
                 double *capacity, double *deltd, double k, double *NumUnit,
                 double *Vof, int *flatbeta, int *indi, int *io, int *Into, int
*Outof,
                 int *ns, int *nd, int *Sour, int *Dest, int *lpstat_p,
                 double *objval_p, double *NumUnitc, double *Daf)

#else
  int
  nt_opt_and_soln (errmsgfunc, handle, sd, ij, node, rout, traffic, capacity,deltd, k,
                   NumUnit, Vof,flatbeta, indi, io, Into, Outof, nd, Sour, Dest,
                   lpstat_p, objval_p, NumUnit,Daf)

  void    (CPXPUBLIC *errmsgfunc)();
  void    *handle;
  int     sd;
  int     ij;
  int     node;
  int     *rout;
  double  *traffic;
  double  *capacity;
  double  *deltd;
  double  k;
  double  *NumUnit;
  double  *Vof;
  int     *flatbeta;
  int     *indi;
  int     *io;
  int     *Into;
  int     *Outof;
  int     *ns;
  int     *nd;
  int     *Sour;
  int     *Dest;
  //double  *delay;

  int     *lpstat_p;
  double  *objval_p;
  double  *NumUnitc;
  double  *Daf;

#endif
{
     char  *probname = "NetworkOp";
     CPXENVptr       env = NULL;
     CPXLPptr        lp  = NULL;
     CPXCHANNELptr   cpxerror   = NULL;
     CPXCHANNELptr   cpxwarning = NULL;

     int     status = 0;
     int     lpstat;
```

```
    double  objval;

    int     numcolsD;     /* for Daf memory sizing, s is a loop counter */

    env = CPXopenCPLEX (&status);

   /* If an error occurs, the status value indicates the  reason for failure.
    * We'll call the errmsgfunc with the erro, because we can't call CPXmsg if
    * CPXopenCPLEXdevelop failed.
    */

    if (env == NULL) {
           char errmsg[1024];
           if (errmsgfunc != NULL) {
                  (*errmsgfunc) (handle, "Could not open CPLEX environment.\n");
                  CPXgeterrorstring (env, status, errmsg);
                  (*errmsgfunc) (handle, errmsg);
        }
        goto TERMINATE;
    }

    status = CPXgetchannels (env, NULL, &cpxwarning, &cpxerror, NULL);
    if (status) {
           if (errmsgfunc != NULL) {
                  (*errmsgfunc) (handle, "Error in CPXgetchannels.\n");
           }
           goto TERMINATE;
    }

    if (errmsgfunc != NULL) {
           status = CPXaddfuncdest (env, cpxerror, handle, errmsgfunc);
           if (!status) {
                  status = CPXaddfuncdest (env, cpxwarning, handle, errmsgfunc);
           }
           if (status) {
                  (*errmsgfunc) (handle, "Error in CPXaddfuncdest.\n");
                  goto TERMINATE;
           }
     }

    /* Now the channels have the error message function, we can use CPXmsg for
errors.
     * Create the problem
     */
     lp = CPXcreateprob (env, &status, probname);

     if (lp == NULL) {
            status = -2;
            CPXmsg (cpxerror, "Failed to create LP.\n");
            goto TERMINATE;
     }

     status = gennt (sd, ij, node, rout, traffic, capacity, deltd, k, NumUnit,Vof,
                     flatbeta, indi, io, Into, Outof, ns, nd, Sour, Dest, env, lp);

     if (status) goto TERMINATE;

     status = CPXlpopt (env, lp);
     if (status) {
            CPXmsg (cpxerror, "Optimization failed. Status %d.\n", status);
            goto TERMINATE;
     }

//CPXsolution (env, lp, &lpstat, &objval, NULL, NULL, NULL, NULL)
     status = CPXsolution (env, lp, &lpstat, &objval, NULL, NULL, NULL, NULL);
     if (status) {
            CPXmsg (cpxerror, "Solution failed. Status %d.\n", status);
            goto TERMINATE;
     }

     *objval_p = objval;
```

```
      *lpstat_p = lpstat;

     /* Now get the solution. Use the fact that the variables are added in
      * the order NumUnitc,Daf, and the sloces of the X vector can be used to copy
over
      * the solution into the flat...arrays.
      */

     /* Get the NumUnit variables */
     //getx(env, lp, double* x, int begin, int end)
     status = CPXgetx (env, lp, NumUnitc, 0, ij-1);
     if (status) {
         CPXmsg (cpxerror, "CPXgetx failed to get NumUnit solution. status %d.\n",
                 status);
         goto TERMINATE;
     }

     /* Get the Daf variables */
     numcolsD = sd*ij;
     status = CPXgetx (env, lp, Daf, ij, ij+numcolsD-1);

     if (status) {
            CPXmsg (cpxerror, "CPXgetx failed to get Daf solution. status %d.\n",
                    status);
            goto TERMINATE;
     }

TERMINATE:
      if (lp != NULL)  CPXfreeprob (env, &lp);

      if (errmsgfunc != NULL && env != NULL) {
             int dfstat = CPXdelfuncdest (env, cpxerror, handle, errmsgfunc);
             if (!dfstat) {
                    dfstat = CPXdelfuncdest (env, cpxwarning, handle, errmsgfunc);
             }
             if (dfstat) {
                    (*errmsgfunc) (handle, "CPXdelfuncdest failed.\n");
             }
             if (dfstat && !status)  status = dfstat;
      }

     /* free up the CPLEX environment, if necessary */
     if (env != NULL) {
            int closestat = CPXcloseCPLEX (&env);

            if (closestat && errmsgfunc != NULL) {
                   char errmsg[1024];
                   (*errmsgfunc) (handle, "Could not close CPLEX environment.\n");
                   CPXgeterrorstring (env, closestat, errmsg);
                   (*errmsgfunc) (handle, errmsg);
            }
            if (closestat && !status) status = closestat;
     }

     return (status);
}  /* End nt_opt_and_soln */
```

7. hnn code

```
import java.io.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.filechooser.*;
import java.beans.*;

class InfraDialog
           extends JDialog
           implements ActionListener
```

```
{
    private JHnn jHnn;
    private JButton okButton;
    private JButton cancelButton;
    private JLabel label1,label2,label3,label4;
    private JButton button1,button2,button3,button4;
    private JFileChooser fileChooser;

    public InfraDialog( JFrame parentFrame ) {
            super( parentFrame );
            jHnn = (JHnn) parentFrame;

            setTitle( "Setup Infrastructure" );
            setDefaultCloseOperation( DISPOSE_ON_CLOSE );

            JPanel topPanel = new JPanel();
            topPanel.setLayout( new GridLayout( 5, 2 ));
            getContentPane().add( topPanel );

            button1 = new JButton(" Traffic Demand ");
            label1 = new JLabel(" N/A      ");
            topPanel.add( button1 );
            topPanel.add( label1 );
            button1.addActionListener(this);

            button2 = new JButton(" Indicator Beta ");
            label2 = new JLabel(" N/A      ");
            topPanel.add( button2 );
            topPanel.add( label2 );
            button2.addActionListener(this);

            button3 = new JButton(" Installed Capacity ");
            label3 = new JLabel(" N/A      ");
            topPanel.add( button3 );
            topPanel.add( label3 );
            button3.addActionListener(this);

            button4 = new JButton(" Rout ");
            label4 = new JLabel(" N/A      ");
            topPanel.add( button4 );
            topPanel.add( label4 );
            button4.addActionListener(this);

           okButton = new JButton("OK");
           cancelButton = new JButton("CANCEL");
           topPanel.add( okButton );
           topPanel.add( cancelButton );

           okButton.addActionListener(this);
           cancelButton.addActionListener(this);

           fileChooser = new JFileChooser();
           fileChooser.setApproveButtonText( "Read into array" );

           this.pack();
    }

      public void actionPerformed( ActionEvent event )
      {
            if( (event.getSource() == okButton) ||
                (event.getSource() == cancelButton) )
            {
                  this.setVisible(false);
                  jHnn.updateStatus();
            }
            if( event.getSource() == button1 )
            {
                 int retval = fileChooser.showOpenDialog(this);
                 if (retval == JFileChooser.APPROVE_OPTION) {
                       jHnn.flow = ParmFileParser.parseSimpleFloatFile(
                                   fileChooser.getSelectedFile() );
```

```
                        jHnn.FlowName = fileChooser.getSelectedFile().getName();
                        label1.setText(jHnn.FlowName + " " +
                                        Integer.toString( jHnn.flow.length ));
                }
            }
            if( event.getSource() == button2 )
            {
                int retval = fileChooser.showOpenDialog(this);
                if (retval == JFileChooser.APPROVE_OPTION) {
                    jHnn.beta = ParmFileParser.parseIntFile(
                                    fileChooser.getSelectedFile() );
                    jHnn.BetaName = fileChooser.getSelectedFile().getName();
                    label2.setText( jHnn.BetaName + " "
                                    + Integer.toString( jHnn.beta.length ) + "x"
                                    + Integer.toString( jHnn.beta[0].length ) );
                }
            }
            if( event.getSource() == button3 )
            {
                int retval = fileChooser.showOpenDialog(this);
                if (retval == JFileChooser.APPROVE_OPTION) {
                    jHnn.C = ParmFileParser.parseSimpleFloatFile(
                                    fileChooser.getSelectedFile() );
                    jHnn.CapName = fileChooser.getSelectedFile().getName();
                    label3.setText(jHnn.CapName + " "
                                    + Integer.toString( jHnn.C.length ));
                }
            }
            if( event.getSource() == button4 )
            {
                int retval = fileChooser.showOpenDialog(this);
                if (retval == JFileChooser.APPROVE_OPTION) {
                jHnn.ROUT = ParmFileParser.parseSimpleIntFile(
                                fileChooser.getSelectedFile() );
                jHnn.RoutName = fileChooser.getSelectedFile().getName();
                label4.setText(jHnn.RoutName + " "
                                + Integer.toString( jHnn.ROUT.length ));
                }
             }
        }
}


import java.io.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.JMenu;
import javax.swing.JMenuItem;
import javax.swing.JMenuBar;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JButton;
import javax.swing.JLabel;
import javax.swing.BorderFactory;
import javax.swing.border.Border;
import javax.swing.border.TitledBorder;
import javax.swing.Box;
import javax.swing.BoxLayout;
import javax.swing.JSplitPane;
import javax.swing.*;
import java.text.*;
public class JHnn extends JFrame
             implements ActionListener
{
    public final static int ONE_SECOND = 300;
    private JButton parmButton;
    private JButton infraButton;
    private JButton simButton;
    private ParmDialog parmDiag;
    private InfraDialog infraDiag;
```

```
  public float  THRESHOLD = (float) 0.001;
  public int    ITERATION = 400;
  public float  ad = (float) 0.06;
  public float  P = (float) 0.4;
  public float  temp = (float) 0.3;
  public float  dt = (float) 0.001;
  public float  a = (float) 0.5;
  public float  c = (float) 0.1;
  public int    inter;
  public int    b = 2300;

  float[]  flow;
  int[][]  beta;
  float[]  C;
  float    totalflow;
  float    totalcap;
  float    maxflow;

  Unit[] array;
  int NPAIR;    // number of (source-destination) node pairs
  int ROUT[];   // number of predetermined routes between (s-d)
  int DLINK;    // number of direct links

  public String FlowName, BetaName, CapName, RoutName;
  JPanel statusPanel;
  public JLabel labelTHRESHOLD,labelITERATION,labelad,labelP;
  public JLabel labeltemp,labeldt,labela,labelb,labelc;
  public JLabel labelFlowName,labelBetaName,labelCapName,labelRoutName;
  public JLabel labelNPAIR,labelDLINK;
  JTextArea resultArea;
  private NumberFormat nf;
  private JRadioButton sigmoidButton, piecewiseButton, routInitButton,
                       sigmoidInitButton;
  private JRadioButton fixedCoButton, floatCoButton;
  private ButtonGroup neuralActionGroup, neuralInitGroup, coefficientGroup;
  private JMenuItem saveResultItem;
  private JProgressBar progressBar;
  private Generate generate;
  private Timer timer;
  private int floatCount;

  //just used for floating coefficient
  private double lastDelta, lastDelay;

public JHnn() {

      //Create the menu bar.
      JMenuBar menuBar = new JMenuBar();
      setJMenuBar(menuBar);
      JMenu m1 = new JMenu("File");
      m1.setMnemonic(KeyEvent.VK_F);
      menuBar.add(m1);
      JMenu m2 = new JMenu("Tool");
      m2.setMnemonic(KeyEvent.VK_T);
      menuBar.add(m2);
      JMenu m3 = new JMenu("Help");
      m3.setMnemonic(KeyEvent.VK_H);
      menuBar.add(m3);

      saveResultItem = new JMenuItem("Save result as", KeyEvent.VK_S);
      saveResultItem.setAccelerator(KeyStroke.getKeyStroke(
                KeyEvent.VK_1, ActionEvent.ALT_MASK));
     saveResultItem.getAccessibleContext().setAccessibleDescription(
                                                "Save result as");
     m1.add( saveResultItem );
    saveResultItem.addActionListener(this);

      JPanel contentPanel = new JPanel();
      setContentPane(contentPanel);

       //A border that puts 10 extra pixels at the sides and bottom of each pane.
```

```
Border paneEdge = BorderFactory.createEmptyBorder(0,10,10,10);
Border paneEdge2 = BorderFactory.createEmptyBorder(10,20,10,20);

FlowName = new String("flow.txt");
flow = ParmFileParser.parseSimpleFloatFile( new File(FlowName) );

BetaName = new String("Beta.txt");
beta = ParmFileParser.parseIntFile( new File(BetaName) );

CapName = new String("Cap.txt");
C = ParmFileParser.parseSimpleFloatFile( new File(CapName) );

RoutName = new String("Rout.txt");
ROUT = ParmFileParser.parseSimpleIntFile( new File(RoutName) );

JPanel buttonPanel = new JPanel();
buttonPanel.setBorder(paneEdge2);
GridBagLayout gridbag = new GridBagLayout();
buttonPanel.setLayout( gridbag );
GridBagConstraints c = new GridBagConstraints();

parmButton = new JButton("Parameter");
c.gridwidth = 2;
c.gridx = 0;
c.gridy = 0;
gridbag.setConstraints(parmButton, c);
buttonPanel.add(parmButton);

infraButton = new JButton("Infrastructure");
c.gridx = 2;
gridbag.setConstraints(infraButton, c);
buttonPanel.add(infraButton);

simButton = new JButton("Neural Network");
c.gridx = 4;
gridbag.setConstraints(simButton, c);
buttonPanel.add(simButton);

parmButton.addActionListener(this);
infraButton.addActionListener(this);
simButton.addActionListener(this);

sigmoidButton = new JRadioButton("sigmoid");
piecewiseButton = new JRadioButton("piecewise");
sigmoidButton.setSelected(true);

routInitButton = new JRadioButton("1/ROUT");
sigmoidInitButton = new JRadioButton("sigmoid");
routInitButton.setSelected(true);

fixedCoButton = new JRadioButton("Fixed");
floatCoButton = new JRadioButton("Float");
fixedCoButton.setSelected(true);

neuralActionGroup = new ButtonGroup();
neuralActionGroup.add(sigmoidButton);
neuralActionGroup.add(piecewiseButton);

neuralInitGroup = new ButtonGroup();
neuralInitGroup.add(routInitButton);
neuralInitGroup.add(sigmoidInitButton);
coefficientGroup = new ButtonGroup();
coefficientGroup.add(fixedCoButton);
coefficientGroup.add(floatCoButton);

c.anchor = GridBagConstraints.WEST;

JLabel actionLabel = new JLabel( "Neuron Actions: " );
c.gridx = 0;
c.gridy = 1;
gridbag.setConstraints(actionLabel, c);
```

```
buttonPanel.add(actionLabel);

c.gridx = 2;
c.gridy = 1;
gridbag.setConstraints(sigmoidButton, c);
buttonPanel.add(sigmoidButton);

c.gridx = 4;
gridbag.setConstraints(piecewiseButton, c);
buttonPanel.add(piecewiseButton);

JLabel initLabel = new JLabel( "Neuron Init: " );
c.gridx = 0;
c.gridy = 2;
gridbag.setConstraints(initLabel, c);
buttonPanel.add(initLabel);

c.gridx = 2;
gridbag.setConstraints(routInitButton, c);
buttonPanel.add(routInitButton);

c.gridx = 4;
gridbag.setConstraints(sigmoidInitButton, c);
buttonPanel.add(sigmoidInitButton);

JLabel coefficientLabel = new JLabel( "Coefficient: " );
c.gridx = 0;
c.gridy = 3;
gridbag.setConstraints(coefficientLabel, c);
buttonPanel.add(coefficientLabel);

c.gridx = 2;
gridbag.setConstraints(fixedCoButton, c);
buttonPanel.add(fixedCoButton);

c.gridx = 4;
gridbag.setConstraints(floatCoButton, c);
buttonPanel.add(floatCoButton);

JPanel statusPanel = new JPanel();
statusPanel.setLayout(new BoxLayout(statusPanel, BoxLayout.Y_AXIS));
Border blackline = BorderFactory.createLineBorder(Color.black);
TitledBorder title1 = BorderFactory.createTitledBorder(blackline, "status");
title1.setTitleJustification(TitledBorder.CENTER);
title1.setTitlePosition(TitledBorder.DEFAULT_POSITION);
statusPanel.setBorder(paneEdge);
addCompForStatusBorder(title1, "This panel is reserved for selected status",
                       statusPanel);

JPanel resultUpperPanel = new JPanel();
GridBagLayout resultGridbag = new GridBagLayout();
resultUpperPanel.setLayout( resultGridbag );
GridBagConstraints resultConst = new GridBagConstraints();
resultConst.anchor = GridBagConstraints.WEST;

JLabel progressLabel = new JLabel( "Progress: " );
resultConst.gridwidth = 1;
resultConst.gridx = 0;
resultConst.gridy = 0;
resultGridbag.setConstraints(progressLabel, resultConst);
resultUpperPanel.add(progressLabel);
generate = new Generate();
progressBar = new JProgressBar(0, generate.getLengthOfTask());
progressBar.setValue(0);
progressBar.setStringPainted(true);
resultConst.gridx = 1;
resultGridbag.setConstraints(progressBar, resultConst);
resultUpperPanel.add(progressBar);
resultArea = new JTextArea();
resultArea.setMargin(new Insets(5,5,5,5));
resultArea.setEditable(false);
```

```
        JScrollPane resultPanel = new JScrollPane();
        resultPanel.getViewport().add( resultArea );
        resultPanel.setBounds( 10, 10, 280, 180 );
        resultArea.append(" \n");
        JPanel resultOutPanel = new JPanel();
        resultOutPanel.setLayout(new BorderLayout());
        resultOutPanel.add(resultUpperPanel,BorderLayout.NORTH);
        resultOutPanel.add(resultPanel,BorderLayout.CENTER);

        //Create a timer.
        timer = new Timer(ONE_SECOND, new ActionListener() {
           public void actionPerformed(ActionEvent evt) {
                if (fixedCoButton.isSelected()) {
                    progressBar.setValue(generate.getCurrent());
                    resultArea.append( generate.getResult() );
                    resultArea.setCaretPosition(resultArea.getDocument().getLength());
                    if (generate.done()) {
                       progressBar.setValue(generate.getCurrent());
                       resultArea.append( generate.getResult() );
                       resultArea.setCaretPosition(
                                                  resultArea.getDocument().getLength());
                       Toolkit.getDefaultToolkit().beep();
                       timer.stop();
                       simButton.setEnabled(true);
                    }
                }//fixedCoButton
                else if (floatCoButton.isSelected()) {
                    progressBar.setValue(generate.getCurrent());
                    resultArea.append( generate.getCoResult() );
                    resultArea.setCaretPosition(resultArea.getDocument().getLength());
                    if (floatCount >= 550) {
                            progressBar.setValue(generate.getCurrent());
                            resultArea.append( generate.getCoResult() );
                            resultArea.setCaretPosition(
                                                  resultArea.getDocument().getLength());
                             Toolkit.getDefaultToolkit().beep();
                             timer.stop();
                             simButton.setEnabled(true);
                    }
                }//floatCoButton
            }
        });

        JSplitPane splitPaneV = new JSplitPane( JSplitPane.VERTICAL_SPLIT);
        JSplitPane splitPaneH = new JSplitPane( JSplitPane.HORIZONTAL_SPLIT);

        splitPaneV.setLeftComponent( buttonPanel );
        splitPaneV.setRightComponent( statusPanel );

        splitPaneH.setLeftComponent( splitPaneV );
        splitPaneH.setRightComponent( resultOutPanel );

        contentPanel.setLayout(new BorderLayout());
        contentPanel.add(splitPaneH,BorderLayout.CENTER);

         parmDiag = new ParmDialog( this );
         infraDiag = new InfraDialog( this );
         nf = NumberFormat.getInstance();

        // Anonymous inner class to terminate program
        this .addWindowListener( new WindowAdapter()
         {
            public void windowClosing( WindowEvent e )
            {
                    System.exit( 0 );
            }
          } ); // End addWindowListener
}

public void actionPerformed( ActionEvent event )
{
```

```
    if( event.getSource() == parmButton )
    {
           parmDiag.setVisible(true);
    }
    if( event.getSource() == infraButton )
    {
           infraDiag.setVisible(true);
    }
    if( event.getSource() == simButton )
    {
         if ( fixedCoButton.isSelected() )
             resultArea.setText( "" );
         else
             resultArea.setText( "b                    c                    load \n" );
         simButton.setEnabled(false);
         progressBar.setValue(progressBar.getMinimum());
         generate.go();
         timer.start();
    }
    if( event.getSource() == saveResultItem )
    {
         JFileChooser saveFileChooser = new JFileChooser();
         int retval = saveFileChooser.showSaveDialog(this);
         if (retval == JFileChooser.APPROVE_OPTION) {
            try {
              FileWriter out = new FileWriter( saveFileChooser.getSelectedFile() );
              out.write( resultArea.getText() );
              out.close();
             }
            catch (FileNotFoundException e1) {
                System.err.println("File not found: ");
            }
            catch (IOException e2) {
                e2.printStackTrace();
             } //end catch
        }
    }
}

void addCompForBorder(Border border, String description, Container container) {
    JPanel comp = new JPanel();
    JLabel label = new JLabel(description, JLabel.CENTER);
    comp.setLayout(new GridLayout(1, 1));
    comp.add(label);
    comp.setBorder(border);

    container.add(Box.createRigidArea(new Dimension(0, 10)));
    container.add(comp);
}

void addCompForStatusBorder(Border border, String description, Container container){
    statusPanel = new JPanel(new GridLayout(11, 2));
    statusPanel.setBorder(border);

    labelTHRESHOLD = new JLabel( " THRESHOLD:    " + Float.toString(THRESHOLD) );
    labelITERATION = new JLabel( " ITERATION:    " + Integer.toString(ITERATION) );
    labeltemp      = new JLabel( " TEMPARATURE:    " + Float.toString(temp) );
    labeldt        = new JLabel( " DELTA T:    " + Float.toString(dt) );
    labela         = new JLabel( " COEFFICIENT A:    " + Float.toString(a) );
    labelb         = new JLabel( " COEFFICIENT B:    " + Integer.toString(b)    );
    labelc         = new JLabel( " COEFFICIENT C:    " + Float.toString(c) );
    labelad        = new JLabel( " MAX AVER DELAY:    " + Float.toString(ad) );
    labelP         = new JLabel( " AVERAGE PACKET SIZE: " + Float.toString(P) );
    labelFlowName  = new JLabel( " Traffic Demand: " + FlowName + " "
                                 + Integer.toString( flow.length ));
    labelBetaName  = new JLabel( " Indicator Beta: " + BetaName + " "
                                 + Integer.toString( beta.length ) + "x"
                                 + Integer.toString( beta[0].length ) );
    labelCapName   = new JLabel( " Installed Capacity: " + CapName + " "
                                 + Integer.toString( C.length ));
    labelRoutName  = new JLabel( " ROUT: " + RoutName + " "
```

```
                                        + Integer.toString( ROUT.length ));

    NPAIR = flow.length;
    DLINK = beta[0].length;

    labelNPAIR     = new JLabel( " NPAIR: " + NPAIR );
    labelDLINK     = new JLabel( " DLINK: " + DLINK );

    statusPanel.add( labelTHRESHOLD );
    statusPanel.add( labelITERATION );
    statusPanel.add( labeltemp );
    statusPanel.add( labeldt );
    statusPanel.add( labela );
    statusPanel.add( labelb );
    statusPanel.add( labelc );
    statusPanel.add( labelad );
    statusPanel.add( labelP );

    statusPanel.add( new JLabel( "    ") );
    statusPanel.add( new JLabel( "-------------------------------") );
    statusPanel.add( new JLabel( "    ") );

    statusPanel.add( labelFlowName );
    statusPanel.add( labelBetaName );
    statusPanel.add( labelCapName );
    statusPanel.add( labelRoutName );

    //statusPanel.add( new JLabel( "    ") );
    statusPanel.add( new JLabel( "-------------------------------") );
    statusPanel.add( new JLabel( "    ") );

    statusPanel.add( labelNPAIR );
    statusPanel.add( labelDLINK );

    //container.add(Box.createRigidArea(new Dimension(0, 10)));
    container.add(statusPanel);
}

public void updateStatus() {
    labelTHRESHOLD.setText( " THRESHOLD:    " + Float.toString(THRESHOLD) );
    labelITERATION.setText( " ITERATION:    " + Integer.toString(ITERATION) );
    labeltemp.setText( " TEMPARATURE:    " + Float.toString(temp) );
    labeldt.setText( " DELTA T:    " + Float.toString(dt) );
    labela.setText( " COEFFICIENT A:    " + Float.toString(a) );
    labelb.setText( " COEFFICIENT B:    " + Integer.toString(b) );
    labelc.setText( " COEFFICIENT C:    " + Float.toString(c) );
    labelad.setText( " LOOK-UP DALAY:    " + Float.toString(ad) );
    labelP.setText( " AVERAGE PACKET SIZE: " + Float.toString(P) );
    labelFlowName.setText( " Traffic Demand: " + FlowName + " "
                                        + Integer.toString( flow.length ));
    labelBetaName.setText( " Indicator Beta: " + BetaName + " "
                                        + Integer.toString( beta.length ) + "x"
                                        + Integer.toString( beta[0].length ) );
    labelCapName.setText( " Installed Capacity: " + CapName + " "
                                        + Integer.toString( C.length ));
    labelRoutName.setText( " ROUT: " + RoutName + " "
                                        + Integer.toString( ROUT.length ));
    NPAIR = flow.length;
    DLINK = beta[0].length;
    labelNPAIR.setText( " NPAIR: " + NPAIR );
    labelDLINK.setText( " DLINK: " + DLINK );
}

public static void main(String[] args) {
      JHnn jhnn = new JHnn();
      jhnn.setTitle("Jia's HNN toolkit");
      jhnn.pack();
      jhnn.setVisible(true);
}

/* f(x) */
```

```
float f(float x)
{
      if (x>=0.0) return x;
      else return (float)0.0;
}  /* end f(x) */

/* random generator -- generate random number from 0-0.1*/
float Rand()
{
   return (float)(Math.random()/10);
}  /* end random generator */

/* allocate memory for units and do initialization */
void alloc_unit(int n)
{
      int i;
      array = new Unit[n];
      for (i=0;i<n;i++)
      {
        array[i] = new Unit();
        array[i].out=0;
        array[i].act=0;
        array[i].bias=0;
        array[i].inhibit=0;
      }
 }

 double piecewise(double i, float temp)
 {
        double val;
        double r_val;

        if ( temp <= 0.0 ) {
           if ( i>0 ) {
              return 1;
           }
           else {
              return 0;
           }
        }
        else val = i / temp;
        return r_val = 1.0 / ( 1.0 + Math.exp( -1.0 * val ) );
 }

 double piecewise1(double i, float temp)
 {
         double val;
         double r_val;

         if ( temp <= 0.0 ) {
             if ( i>0 ) {
                 return 1;
                 }
                 else {
                         return 0;
                 }
     }
         else val = i / temp;

         if (val>1) return 1.0;
         if (val<0) return 0.0;
         else return val;
 }

 int getArrayMax(int[] inputArray) {
         int max = 0;

         for (int i=0; i < inputArray.length; i++) {
                 if ( inputArray[i] > max ) {
                         max = inputArray[i];
                 }
```

```
        }
        return max;
}

float getArrayMaxf(float[] inputArray) {
                float max = 0;

                for (int i=0; i < inputArray.length; i++) {
                        if ( inputArray[i] > max ) {
                                max = inputArray[i];
                        }
                }
                return max;
}

float getTotalArray(float[] inputArray) {
        float total = 0;

        for (int i=0; i < inputArray.length; i++) {
                total += inputArray[i];
        }
        return total;
}

/* calculate the whole flow for an optimal output */
double eflow()
{
        int i,j,k,row;
        double load, fl=0.0;

        for (k=0;k<DLINK;k++){
                load=0.0;
                row = 0;
                for (i=0; i<NPAIR; i++)
                for (j=0;j<ROUT[i];j++)
                {
                  load+=flow[i]*beta[row][k]*array[row].act;
                  row++;
                } /* end i,j */
                //fl+=(load/(C[k]-load)+ad*load/P)/(1000*P);
                fl +=load;
        } /* end k */
        return fl;
}   /* end of eflow */

String formatDouble(double i) {
        nf.setMaximumFractionDigits(5);
        nf.setMinimumFractionDigits(5);
        return nf.format(i);
}

public class Generate {
    private StringBuffer result, coresult;
    private int lengthOfTask;
    int current = 0;
    private int stringPosition = 0;
    private int costringPosition = 0;
    int row = 0;

    Generate() {
        lengthOfTask = 1000;
        result = new StringBuffer();
        coresult = new StringBuffer();
    }

    /**
     * Called from JHnn event dispatch thread to start the task.
     */
    void go() {
        current = 0;
        stringPosition = 0;
```

```
        costringPosition = 0;
        result.delete( 0, result.length() );
        coresult.delete( 0, coresult.length() );
        final SwingWorker worker = new SwingWorker() {
            public Object construct() {
                return new GenerateNetwork();
            }
        };
        worker.start();
    }

     /**
     * Called from JHnn to find out how much work needs
     * to be done.
     */
    int getLengthOfTask() {
        return lengthOfTask;
    }

    /**
     * Called from JHnn to find out how much has been done.
     */
    int getCurrent() {
        return current;
    }

    void stop() {
        current = lengthOfTask;
    }

    /**
     * Called from JHnn to find out if the task has completed.
     */
    boolean done() {
        if (current >= lengthOfTask)
            return true;
        else
            return false;
    }

    // just return the difference since last getResult
    String getResult() {
              String resultString;
              if ( stringPosition < result.length() ) {
                      resultString = result.substring( stringPosition );
                      stringPosition = result.length();
                  }
                  else {
                      resultString = "";
                  }
              return resultString;
    }

    // just return the difference since last getCoResult
    String getCoResult() {
              String coresultString;
              if ( costringPosition < coresult.length() ) {
                      coresultString = coresult.substring( costringPosition );
                      costringPosition = coresult.length();
               }
               else {
                       coresultString = "";
                }
                return coresultString;
}

/* generate the network */
class GenerateNetwork {
      GenerateNetwork()
       {
            int i,j,k;
```

```
                double wght[] = new double[NPAIR];

                double   delta;
                double   ua, ub, uc;
                float[]  dd = new float[DLINK];
                int it,m,n,iter;

                floatCount = 0;

                do {
                //initialize
                       delta = 0;
                       iter = 0;
                       current = 0;
                       stringPosition = 0;
                       result.delete( 0, result.length() );

                /* allocate space for units */
                       alloc_unit( NPAIR * getArrayMax(ROUT) );

/* store the total flow for each node pair*beta of each direct link in .bias */
                       row = 0;
                       for (i=0;i<NPAIR;i++)
                         for (j=0;j<ROUT[i];j++) {
                           for (k=0;k<DLINK;k++)
                           {
                              array[row].bias += flow[i]*beta[row][k];
                           }
                           row++;
                         }

                      maxflow = getArrayMaxf(flow);
                      totalcap = getTotalArray(C);
                      totalflow = getTotalArray(flow);

           //wght=temp/ROUT;
           for (i=0;i<NPAIR;i++) {
                       wght[i] = -0.5 * temp * Math.log( (double)(ROUT[i]) );
           }

           row = 0;
           for (i=0;i<NPAIR;i++)
               for (j=0;j<ROUT[i];j++)  {
                       /*sigmoid initialization */
                       array[row].excit=wght[i]*(1.0+Rand());
                       array[row].act=piecewise(array[row].excit, temp);

                       /* average initialization */
                       //array[row].act=1 / ROUT[i]*(1.0+Rand());

                       row++;
            }  /* end  */

           result.append("   bias      excit     act      \n");

           row = 0;
           for (i=0;i<NPAIR;i++)
              for (j=0;j<ROUT[i];j++) {
                  result.append( formatDouble(array[row].bias) + "   "
                               + formatDouble(array[row].excit) + "   "
                               + formatDouble(array[row].act) + "\n" );
                row++;
           }
           result.append("\n");

           current = 100;

           /* simulate */
           for (it=0; it<ITERATION; it++)
           {
                result.append( "iteration time:" + it + "\n" );
```

```
                    iter++;
                    current += (int) ( 900 / ITERATION ); // updating progress bar
                    Thread.yield();
                     /* calculate the differential of energy function for time t */
                    result.append( "---calculate the differential of energy function
for time t ---\n" );
                    result.append("  ua        ub       uc        \n");

                    row = 0;
                    int row1 = 0;
                    for(i=0; i<NPAIR; i++)  {
                           ub=0;
                           for (j=0; j<ROUT[i]; j++) {
                           /* the second term */
                                ub += array[row1].act;
                                row1 ++;
                           }

                    for(j=0, ua=0, uc=0; j<ROUT[i]; j++)  {
                           /* the first term */
                           ua = array[row].bias;

                           /* the third term */
                           for(k=0; k<DLINK; k++){
                              dd[k]=0;
                              int row2 = 0;
                              for(m=0; m<NPAIR; m++)
                                 for(n=0; n<ROUT[m]; n++) {
                                    dd[k]+=flow[m]*beta[row2][k]*array[row2].act;
                                    row2++;
                                 }
                              uc +=f(dd[k]+P*8/ad-C[k]);  //+P/ad
                           } //end k

                    result.append( formatDouble(ua) + "    "
                           + formatDouble(ub) + "    " + formatDouble(uc) + "\n" );

                    /* update the total input to a unit */
                    array[row].excit -= dt * ( array[row].excit
                                        +a*ua + 2*b*(ub-1) + 2*c*uc);
                    row++;
                  } //end j
                 }//end i
                    result.append("\n");

                    /* update activation in paralell */
                    result.append( "---update activation in paralell---\n" );
                    result.append("  bias       excit      acts       \n");
                    row = 0;
                    for (i=0;i<NPAIR;i++) {
                        for (j=0;j<ROUT[i];j++) {
                           array[row].act=piecewise(array[row].excit, temp);
                           result.append( formatDouble(array[row].bias) + "    "
                                        + formatDouble(array[row].excit) + "    "
                                        + formatDouble(array[row].act) + "\n" );
                           row++;
                         }
                    } //end i
                    result.append("\n");

                    /* calculate the delta in an iteration */
                    result.append( "---calculate the delta in an iteration---\n" );
                    delta=0.0;
                    row = 0;
                    for (i=0;i<NPAIR;i++) {
                         for (j=0;j<ROUT[i];j++) {result.append("inhibit:" +
                                        formatDouble(array[row].inhibit) + "  " );
                         delta += Math.abs( array[row].act - array[row].inhibit );
                         //result.append( "delta:" + formatDouble(delta) + "\n" );
                          row++;
                         }
```

```
                    }//end i
                    result.append( "delta:" + formatDouble(delta) + "\n" );
                     result.append("\n");

                    /*copy the activation level to array[].inhibit */
                    result.append( "---copy the activation level to array[].inhibit---
                                    \n" );
                    row = 0;
                    for (i=0;i<NPAIR;i++) {
                         for (j=0;j<ROUT[i];j++){
                            array[row].inhibit=array[row].act;
                            result.append( formatDouble(array[row].inhibit) );
                            row++;
                         }
                         result.append("\n");
                    }

                    result.append("\n");
                    result.append( "delta is " + formatDouble(delta) + "\n" );

                    if (delta<THRESHOLD) break;
                    } // end it

                    //used for floating coefficient
                    lastDelta = delta;
                    result.append("\n");
                    float[] sum = new float[NPAIR];
                    row = 0;
                    for(i=0;i<NPAIR;i++) {
                            for(j=0;j<ROUT[i];j++) {
                                    sum[i] += array[row].act;
                                    row ++;
                            }
                    }
                    result.append("---final distribution---\n");
                    row = 0;
                    for(i=0;i<NPAIR;i++) {
                       for(j=0;j<ROUT[i];j++) {
                         result.append( formatDouble(array[row].act/sum[i]) + "  " );
                         row++;
                        }
                     result.append("\n");
                     } // end i
                    result.append("\n");

                    //lasteflow used for floating coefficient
                    lastEflow = eflow();
                    result.append( Integer.toString(iter)
                                   + " route iteration, total eflow is: "
                                    + formatDouble(lastEflow) + "\n" );
                    coresult.append( Integer.toString(b));
                    coresult.append( "                  " + Float.toString(c));
                    coresult.append( "          " + formatDouble(lastEflow) + "\n");
                    floatCount++;
                    c =(float)(0.1 + (Math.round(Math.random())*100/1000);
                    inter = (int)(2*a*maxflow + c*(4*totalflow+DLINK*P*8/ad-totalcap));
                    b =(int)(inter + 10 * Math.round(Math.random()));

                    if (fixedCoButton.isSelected()) {
                            current = 1000; //finish the generation
                           break;
                     }
                  } while (floatCount<inter || floatCoButton.isSelected());
               } // end of GenerateNetwork constructor
            }//end of class GenerateNetwork
    } // end of class Generate
}


import java.io.*;
import java.awt.*;
```

```
import java.awt.event.*;
import javax.swing.*;

class ParmDialog
            extends JDialog
            implements ActionListener
{
    private JHnn jHnn;
    private JButton okButton;
    private JButton cancelButton;
    private JTextField field1,field2,field3,field4,field5,field6,field7,field8,field9;

    public ParmDialog( JFrame parentFrame ) {
            super( parentFrame );
            jHnn = (JHnn) parentFrame;
            setTitle( "Set Parameters" );
            setDefaultCloseOperation( DISPOSE_ON_CLOSE );
            JPanel topPanel = new JPanel();
            topPanel.setLayout( new GridLayout( 10, 2 ));
            getContentPane().add( topPanel );

            field1 = new JTextField(Float.toString( jHnn.THRESHOLD ), 8 );
            field2 = new JTextField(Integer.toString( jHnn.ITERATION ), 8 );
            field3 = new JTextField(Float.toString( jHnn.temp ), 8 );
            field4 = new JTextField(Float.toString( jHnn.dt ), 8 );
            field5 = new JTextField(Float.toString( jHnn.a ), 8 );
            field6 = new JTextField(Integer.toString( jHnn.b ), 8 );
            field7 = new JTextField(Float.toString( jHnn.c ), 8 );
            field8 = new JTextField(Float.toString( jHnn.ad ), 8 );
            field9 = new JTextField(Float.toString( jHnn.P ), 8 );

            JLabel label1 = new JLabel( " THRESHOLD:            " );
            JLabel label2 = new JLabel( " ITERATION:            " );
            JLabel label3 = new JLabel( " TEMPARATURE:          " );
            JLabel label4 = new JLabel( " DELTA T:              " );
            JLabel label5 = new JLabel( " COEFFICIENT A:        " );
            JLabel label6 = new JLabel( " COEFFICIENT B:        " );
            JLabel label7 = new JLabel( " COEFFICIENT C:        " );
            JLabel label8 = new JLabel( " LOOK-UP DALAY:        " );
            JLabel label9 = new JLabel( " AVERAGE PACKET SIZE:  " );

            label1.setLabelFor( field1 );
            label2.setLabelFor( field2 );
            label3.setLabelFor( field3 );
            label4.setLabelFor( field4 );
            label5.setLabelFor( field5 );
            label6.setLabelFor( field6 );
            label7.setLabelFor( field7 );
            label8.setLabelFor( field8 );
            label9.setLabelFor( field9 );

            topPanel.add( label1 );
            topPanel.add( field1 );
            topPanel.add( label2 );
            topPanel.add( field2 );
            topPanel.add( label3 );
            topPanel.add( field3 );
            topPanel.add( label4 );
            topPanel.add( field4 );
            topPanel.add( label5 );
            topPanel.add( field5 );
            topPanel.add( label6 );
            topPanel.add( field6 );
            topPanel.add( label7 );
            topPanel.add( field7 );
            topPanel.add( label8 );
            topPanel.add( field8 );
            topPanel.add( label9 );
            topPanel.add( field9 );

            okButton = new JButton("OK");
```

```
            cancelButton = new JButton("CANCEL");
            topPanel.add( okButton );
            topPanel.add( cancelButton );
            okButton.addActionListener(this);
            cancelButton.addActionListener(this);
            this.pack();
    }

     public void actionPerformed( ActionEvent event )
      {
              if( event.getSource() == okButton )
              {
                      jHnn.THRESHOLD = Float.parseFloat( field1.getText() );
                      jHnn.ITERATION = Integer.parseInt( field2.getText() );
                      jHnn.temp = Float.parseFloat( field3.getText() );
                      jHnn.dt = Float.parseFloat( field4.getText() );
                      jHnn.a = Float.parseFloat( field5.getText() );
                      jHnn.b = Integer.parseInt( field6.getText() );
                      jHnn.c = Float.parseFloat( field7.getText() );
                      jHnn.ad = Float.parseFloat( field8.getText() );
                      jHnn.P = Float.parseFloat( field9.getText() );

                      jHnn.updateStatus();
                      this.setVisible(false);
              }
              if( event.getSource() == cancelButton )
              {
                      this.setVisible(false);
              }
      }
}


import java.io.*;
import java.util.*;

public class ParmFileParser {
    public static int[][] parseIntFile(File parmFile) {
        int[] parmLine;
        Vector parmVector = new Vector();
        try {
                    // Create a buffered reader to read each line from parm file
                    BufferedReader in = new BufferedReader(new FileReader(parmFile));
                    String s = in.readLine();
                    while ( s != null ) {
                            StringTokenizer st = new StringTokenizer(s);
                            parmLine = new int[st.countTokens()];
                            int i = 0;
                            while (st.hasMoreTokens()) {
                             parmLine[i] = Integer.valueOf(st.nextToken()).intValue();
                             i++;
                            }
                            parmVector.add(parmLine);
                            s = in.readLine();
                      }
                      in.close();
            } catch (FileNotFoundException e1) {
                    System.err.println("File not found: ");
            } catch (IOException e2) {
                    e2.printStackTrace();
            }

        int[][] parm = new int[parmVector.size()][];
        for (int i = 0; i < parm.length; i++) {
                    parm[i] = (int[])(parmVector.get(i));
        }
        return parm;
    }

    public static float[] parseSimpleFloatFile(File parmFile) {
        Vector parmVector = new Vector();
```

```
        try {
                    // Create a buffered reader to read each line from parm file
                    BufferedReader in = new BufferedReader(new FileReader(parmFile));
                    String s = in.readLine();
                    while ( s != null ) {
                            StringTokenizer st = new StringTokenizer(s);
                            while (st.hasMoreTokens()) {
                             String temp = st.nextToken();
                             parmVector.add( Float.valueOf(temp) );
                            }
                            s = in.readLine();
                    }
                    in.close();
            } catch (FileNotFoundException e1) {
                    System.err.println("File not found: ");
            } catch (IOException e2) {
                    e2.printStackTrace();
            }

        float[] parm = new float[parmVector.size()];
        for (int i = 0; i < parm.length; i++) {
                    Float element = (Float) parmVector.get(i);
                    parm[i] = element.floatValue();
         }
         return parm;
    }

    public static int[] parseSimpleIntFile(File parmFile) {
        Vector parmVector = new Vector();
        try {
                    // Create a buffered reader to read each line from parm file
                    BufferedReader in = new BufferedReader(new FileReader(parmFile));
                    String s = in.readLine();
                    while ( s != null ) {
                            StringTokenizer st = new StringTokenizer(s);
                            while (st.hasMoreTokens()) {
                             String temp = st.nextToken();
                             parmVector.add( Integer.valueOf(temp) );
                            }
                            s = in.readLine();
                     }
                   in.close();
        } catch (FileNotFoundException e1) {
                    System.err.println("File not found: ");
        } catch (IOException e2) {
                    e2.printStackTrace();
        }

        int[] parm = new int[parmVector.size()];
        for (int i = 0; i < parm.length; i++) {
                    Integer element = (Integer) parmVector.get(i);
                    parm[i] = element.intValue();
         }
         return parm;
    }
}

public class Unit {
    public char      state;              // active, inert: 1,0
    public char      type;               // input, output or other
    public double    act;                // activation
    public double    out;                // output, maybe different from act
    public double    bias;               // bias of the unit
    public double    inhibit;            // total inhibit input
    public double    excit;              // total excitatory inout
}


import javax.swing.SwingUtilities;

/**
```

```
* This is the 3rd version of SwingWorker (also known as
* SwingWorker 3), an abstract class that you subclass to
* perform GUI-related work in a dedicated thread.  For
* instructions on using this class, see:
*
* http://java.sun.com/docs/books/tutorial/uiswing/misc/threads.html
*
* Note that the API changed slightly in the 3rd version:
* You must now invoke start() on the SwingWorker after
* creating it.
*/
public abstract class SwingWorker {
    private Object value;  // see getValue(), setValue()
    private Thread thread;

    /**
     * Class to maintain reference to current worker thread
     * under separate synchronization control.
     */
    private static class ThreadVar {
        private Thread thread;
        ThreadVar(Thread t) { thread = t; }
        synchronized Thread get() { return thread; }
        synchronized void clear() { thread = null; }
    }

    private ThreadVar threadVar;

    /**
     * Get the value produced by the worker thread, or null if it
     * hasn't been constructed yet.
     */
    protected synchronized Object getValue() {
        return value;
    }

    /**
     * Set the value produced by worker thread
     */
    private synchronized void setValue(Object x) {
        value = x;
    }

    /**
     * Compute the value to be returned by the <code>get</code> method.
     */
    public abstract Object construct();

    /**
     * Called on the event dispatching thread (not on the worker thread)
     * after the <code>construct</code> method has returned.
     */
    public void finished() {
    }

    /**
     * A new method that interrupts the worker thread.  Call this method
     * to force the worker to stop what it's doing.
     */
    public void interrupt() {
        Thread t = threadVar.get();
        if (t != null) {
            t.interrupt();
        }
        threadVar.clear();
    }

    /**
     * Return the value created by the <code>construct</code> method.
     * Returns null if either the constructing thread or the current
     * thread was interrupted before a value was produced.
```

```
     *
     * @return the value created by the <code>construct</code> method
     */
    public Object get() {
        while (true) {
            Thread t = threadVar.get();
            if (t == null) {
                return getValue();
            }
            try {
                t.join();
            }
            catch (InterruptedException e) {
                Thread.currentThread().interrupt(); // propagate
                return null;
            }
        }
    }

    /**
     * Start a thread that will call the <code>construct</code> method
     * and then exit.
     */
    public SwingWorker() {
        final Runnable doFinished = new Runnable() {
           public void run() { finished(); }
        };

        Runnable doConstruct = new Runnable() {
            public void run() {
                try {
                    setValue(construct());
                }
                finally {
                    threadVar.clear();
                }
                SwingUtilities.invokeLater(doFinished);
            }
        };
        Thread t = new Thread(doConstruct);
        threadVar = new ThreadVar(t);
    }

    /**
     * Start the worker thread.
     */
    public void start() {
        Thread t = threadVar.get();
        t.setPriority(Thread.NORM_PRIORITY-1);
        if (t != null) {
            t.start();
        }
    }
}
```