OPTIMAL QUASI-STATIC ROUTING FOR VIRTUAL CIRCUIT NETWORKS SUBJECTED TO STOCHASTIC INPUTS

by

Wei Kang Tsai

S.B. Massachusetts Institute of Technology

(1979)

S.M. Massachusetts Institute of Technology

(1982)

E.E. Massachusetts Institute of Technology

(1982)

SUBMITTED IN PARTIAL FULFILLMENT OF THE

REQUIREMENTS FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June, 1986

Signature of Author . . . .
Department of Electrical Engineering and Computer Science
June, 2, 1986

Certified by . . . .
Dimitri P. Bertsekas
Thesis Supervisor

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

OPTIMAL QUASI-STATIC ROUTING FOR VIRTUAL CIRCUIT NETWORKS SUBJECTED TO STOCHASTIC INPUTS

by

Wei Kang Tsai

Submitted to the Department of Electrical Engineering and Computer Science on June 2, 1986 in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

ABSTRACT

This research evaluates the performances of two classes of quasi-static routing methods for virtual circuit data networks–adaptive shortest path and gradient projection–in a stochastic asynchronous environment. The routing models take into account asynchronous updates, delays in forwarding control messages, flow transients and violation of the quasi-static load assumption. The congestion measure assumed is a convex increasing separable cost function of total link flows. The optimal cost is the minimal cost obtainable when the load equals its long term average.

A deterministic example which shows that shortest path routing with link length functions taking values from a discrete set does not converge to a long-term optimal routing is presented.

For the gradient projection routing method with randomized path assignment during routing intervals, a new convergence result is obtained. It is shown that the expected cost decreases to a neighborhood of the optimal cost at an exponential rate which is lower bounded by the slowest virtual circuit departure rate. When the stepsize is small enough, the size of the neighborhood is "proportional" to the extent of violation of a so-called "many small users" assumption.

An open problem for the gradient projection routing method is assigning paths for incoming virtual circuits so as to achieve the desired path flows which are determined at the latest routing update. It is shown that, by assigning virtual circuits to paths which are most deficient in the desired number of virtual circuits, a similar convergence result holds.

Digital simulation results using more realistic routing models confirm the theory and provide additional insights.

Thesis Supervisor: Dr. Dimitri P. Bertsekas

Title: Professor of Electrical Engineering

Acknowledgements

I would like to thank my thesis advisor, Professor Dimitri Bertsekas, for his guidance, support and encouragement. His insight and approach of conducting a research helped shape my professional development.

Thanks also to Professor John Tsitsiklis who spent much time discussing the technical details of this thesis, and provided invaluable insight. I would also like to thank Professor Robert Gallager for serving as a thesis reader and sharing his understanding on the subject of data networks.

I am very grateful to Professors George Verghese and Bernard Levy. I appreciate their guidance as teachers and friends. I would like to thank Professors Fred Scheweppe and Leonard Gould for their friendship and support.

I will not forget my office-mates who spent much time with me in discussions and brain-storming sessions. I would like to thank them all : Réne Cruz, Ellen Hahne, Patrick Hosein, Atul Khanna, Jay Kuo, Whay Lee, Utpal Mukherji, Jean Régnier, and John Spinelli. Their encouragement, criticism, and friendship greatly enriched my graduate studies at MIT. In particular, I would like to thank Jay whose friendship is deeply appreciated. In addition, I have to thank Alain Cohen and Steve Baraniuk who coded the OPNET simulator and helped set up my simulation.

This work has been supported in part by the National Science Foundation under grant NSF/ECS 82-17668.

Finally, I owe my deep appreciation and gratitude to my wife, Mei-Huei, my parents, Yu-Chi Tsai and Hwa-Chu Tsai, and my brother Wei-Tek Tsai. Their unfailing love and faith in me sustained me through the difficult times. In addition, many thanks to my wife Mei-Huei for her excellent typing and proof-reading of the thesis.

To God, my Savior and Lord, and my family I dedicate this thesis.

CONTENTS

List of Figures

CHAPTER ONE
INTRODUCTION

1.1 Problem Statement

There is a dilemma in the research on routing in data networks. On one hand, the practical operating environment for routing is so complex [Kobayashi and Konheim 1977] that no mathematically tractable model can adequately describe the physical phenomenon. Thus most analytical studies use either a deterministic nonlinear programming formulation (e.g., [Fratta, Gerla and Kleinrock 1973]) or a steady-state analysis on a simplified synchronous model (e.g., [Yum and Schwartz 1981; Boorstyn and Livne 1981]). On the other hand, simulation studies which may be capable of realistically modeling the physical routing environment cannot provide a clear direction to improve the routing performance.

Two recent studies by Gafni and Bertsekas [1983b] (referred to as [GB]) and Tsitsiklis and Bertsekas [1986] (referred to as [TB]) make an important contribution to the analytical study on practical routing. [GB] shows that, in a *stochastic* routing environment, the *synchronous shortest path* routing converges to a neighborhood of an optimal routing. If the updating period is relatively small as compared to the average virtual circuit holding time, the size of the neighborhood tends to zero as the extent of violation of the quasi-static assumption tends to zero. [TB] shows that a class of distributed optimal *gradient projection* routing methods converges , despite the delay in control packets, measurement inaccuracy, and asynchronous updating. This work is the first to study optimal routing with an explicit *distributed asynchronous* model.

The natural extension of [GB] and [TB] is to combine their routing models to study both the *shortest path* and the *gradient projection* methods. Both classes of methods are important theoretically and practically. The purpose of this thesis research is to study these two classes of routing methods using a *distributed asynchronous stochastic* model. The shortest path method is chosen mainly because of its popularity; the gradient projection method is studied due to its many advantages over other types of algorithms [Bertsekas 1980; Bertsekas 1982d]. We shall focus more on the gradient projection method in this thesis.

We assume a long-haul wire virtual circuit data network in which communication resources are scarce relative to computation resources, and there are no multiple-access media. Generally, some simplifying assumptions have to be made in any analytical study on optimal routing. Typical assumptions are as follows.

(A) The Quasi-Static Load Assumption states that the external traffic arrival rate for each origin-destination (OD) pair is slowly changing over time. This is an approximately valid description of a situation where the rate at which the offered load statistics change is reasonably slower than the convergence rate of the routing algorithm. Usually individual offered traffic samples are assumed not to exhibit frequently large and persistent deviations from their averages. Physically such a situation arises when there is a large number of user-pair conversations associated with each OD pair, and each of these conversations has a traffic rate that is small relative to the total traffic rate for the OD pair. This is the so-called *many small users* environment.

(B) The Fast Settling Time Assumption states that the link (or path) flows (measured in data unit/time unit) adjust to the desired flows instantly at a routing update. Hence the transient of the flows in a routing change is negligible. This assumption is approximately valid in datagram networks but less so in virtual circuit (VC) networks where VC re-routings are forbidden or infrequent.

(C) The Synchronous Update Assumption states that, in a distributed routing, all link flows are measured simultaneously and this information is received in time to carry out

a simultaneous routing update at all network nodes responsible for routing. In a centralized routing algorithm, such an assumption is usually valid. However there are disadvantages such as unreliability in such schemes. Since technical reasons, such as software complexity, are against enforcing a synchronous update protocol, this assumption is usually not valid. An example is the new ARPANET routing algorithm [McQuillan et. al. 1980].

(D) Perfect Measurement Assumption states that link or path flows, and perhaps incremental link costs also, can be measured exactly. It is generally difficult to measure the rate of bursty random data flow in a link operating with data link control. This assumption is an idealization that is usually made to keep the analysis tractable. Segall [1977] considers the problem of estimating link incremental costs.

Most of the existing analyses take for granted the above assumptions. In contrast, we shall develop a routing model, by combining the models in [GB] and [TB], in which the above four assumptions are dispensed with. Our model will explicitly describe how these assumptions are violated. However, the following commonly held assumptions in a quasi-static routing study are retained:

(E) Conservation of Flow Assumption states that the traffic into a node for a given destination is equal to the traffic out of the node for that destination. This assumption is good when the routing update interval, i.e., the time between two consecutive routing updates, is large as compared to the queueing time-constants.

(F) Convex Separable Cost Assumption states that a reasonable convex increasing cost can be assigned to each link as a function of the total link flow, and the objective of routing is to minimize the sum of such costs.

(G) Fixed Network Topology Assumption states that during the time period of interest no topological changes occur. In other words, the routing algorithm will not be

responsible for failure recovery and addition of nodes and links.

Our routing model which is constructed on the basis of the above assumptions can be formulated as a stochastic multicommodity flow problem similar to that of Bertsekas [1980]. This formulation uses path flows as the routing variables, different from the link flow counterpart formulation of Gallager [1977]. The path flow formulation is more natural for virtual circuit oriented routing which is used by most of the operating data networks in one form or another [Schwartz and Stern 1980]. Comparison of the two formulations is briefly given in Bertsekas [1982d] and Bertsekas et. al. [1984].

The objective of this thesis research is to evaluate the performances of the two classes of routings in terms of the cost assumed in *(E)* with the stochastic asynchronous routing model we develop. From the viewpoint of adaptive routing, a good algorithm should not require the updating period to be very small since routing update is expensive. It should be adapted to current loads and the routing variables should be easily computable. From the viewpoint of stochastic optimization, we can evaluate these algorithms using the same criteria such as simple stepsize rules, automatic scaling factors, convergence, fast convergence rates, stable convergence behavior and so on.

So far our major concern on routing algorithms is that of computing routing updates. There is another problem, however, of implementing these routing updates during the time intervals between two consecutive updates. We are interested in this problem because the implementation of routing updates determines the transient behavior of flows, which consequently affects the overall performance of the routing algorithm. We shall refer to this problem as the the *path assignment* problem which is described as follows.

Each path of every OD pair is modeled as a queue with an infinite number of identical exponential servers and VCs as customers. The VCs are assumed to arrive at the origin node of each OD pair at respective Poisson rates. The problem is, given initial flows at the beginning of an updating interval, for each origin node to assign the forthcoming VCs to the respective paths so that the path flows at the end of the updating interval will be as close as possible to the desired flows . The closeness is measured by some cost functions. We also assume that the assignment of VCs is made according to the state of queues at each arrival of a VC, hence the problem is a special case of dynamic routing problem.

Two extreme methods are possible–assigning the VCs randomly with fixed probabilities or assigning the VCs with some deterministic threshold rules. We call the first method *randomization* and the second method *metering.* It is generally believed that metering performs better than randomization with respect to some criteria; however no proof has been offerred to our knowledge.

The difficulty involved in such a study is that even the simplest dynamic routing scheme can lead to queue behavior whose statistical characteristics are either hard to calculate or not yet adequately understood (see the comments by Ephremides, Varaiya and Walrand [1980] and Sarachik [1984]).

To conclude, the result of this thesis research should contribute to the designing and evaluating of routing algorithms, as well as further understanding of the two classes of stochastic optimization algorithms.

1.2 Summary Of Previous Works

1.2.1 Quasi-Static Routing Models

Routing algorithms can be roughly classified as *static, quasi-static* and *dynamic,* according to the time-frame best describing the routing dynamics. The concept of quasi-static routing can be attributed to Gallager [1977].

Unlike a static routing algorithm where the route computations can be done off-line, a quasi-static routing algorithm operates on-line. From time to time, the measurement of current traffic parameters is taken and is sent to the nodes responsible for routing calculations to be used in performing one or more iterations of the algorithm. The algorithm is usually based on a static model with stationary traffic and unchanging network. If the inputs were fixed, the algorithm would converge to a deterministic optimal routing; if the inputs were slow-varying, the algorithm was hoped to *track* the variations and keep the routing close to the changing optimum.

The static optimal routing problem on which optimal quasi-static routing is based was first formulated by Frank and Chou [1971] as a multicommodity flow problem. The most common objective function, the average delay, was due to Kleinrock [1964]. Gallager's [1977] formulation is an improved version of the earlier formulations based on link flows. Bertsekas [1980] first formulated a multicommodity flow problem based on path flows.

To capture the effects of slow varying dynamics in quasi-static routing, dynamic models are called for. The first stochastic model, given by Segall [1977], follows link flow formulation. Gafni and Bertsekas [1983b] explicitly account for the *many small users* routing environment and provide a stochastic model based on path flows. Their earlier paper [1983a], however, provides a deterministic model using flow approximation. A contribution of Gafni and Bertsekas [1983b] is that they quantify the degree of violation of the *many small users* assumption (defined in section 2.2) with a single parameter and, as a result, a form of the law of large number becomes evident.

Distributed asynchronous routing is a special case of general distributed asynchronous algorithms. Bertsekas, Tsitsiklis and Athans [1984] provide an extensive survey of current research results in this field. The routing model we pattern after is that of Tsitsiklis and Bertsekas [1985] which evolved from Tsitsiklis [1984], and earlier works by Bertsekas

[1982b, 1983]. This model is general enough to encompass a large class of update protocols-protocols that are distributed or centralized, synchronous or asynchronous. This model, when combined with that of Gafni and Bertsekas [1983b], is more realistic than most of the existing models since it does away with assumptions $(A) - (D)$.

1.2.2 Algorithms for Quasi-Static Routing

Any nonlinear programming algorithm that solves the multicommodity flow problem, with the property that every one or more iterations constitute a descent step, is a potential candidate for quasi-static routing algorithm. Indeed, a large number of quasi-static routing algorithms can be found in the literature [Schwartz and Stern, 1980].

First is the popular class of shortest path methods. They are motivated by the well-known optimality condition: a routing is optimal if and only if it routes the traffic exclusively on shortest paths where, for each link, the link length is the first derivative of the cost function with respect to the total link flow (e.g., [Gallager 1977]). Most of the these methods are heuristic [Schwartz and Stern 1980] and have been known to be either strictly suboptimal or to exhibit oscillatory behaviors [Bertsekas 1982b].

In a virtual circuit network without VC re-routing, the shortest path method resembles the Flow Deviation (FD) method which is first proposed by Fratta, Gerla and Kleinrock [1973]. The FD method is in essence a Frank-Wolfe method for nonlinear programming, and thus suffers from slow sublinear convergence rate (e.g., [Dunn 1979]).

While shortest path routing algorithms with Lipschitz continuous link lengths have been shown, by Gafni and Bertsekas [1983b], to converge asymptotically under certain assumptions, it is yet unknown that if such algorithms will converge or not when the link lengths take values from a discrete set.

Second is the class of routing algorithms which is based on a class of gradient projection methods for nonlinear programming. This class of nonlinear programming methods includes the original proposal by Rosen, the reduced gradient methods, the convex simplex method and etc. (see Luenberger [1973]). While these methods are attractive for small-scale problems, they are highly unsuitable for large-scale problem with many constraints (see the discussion in the papers by Bertsekas [1976, 1982a] and Bertsekas and Gafni [1983]). Earlier, a routing algorithm based on this class was proposed by Schwartz and Cheung [1976]. Their computational results show that their method is unsuitable for a large number of commodities.

Along with introducing the concept of quasi-static routing, Gallager [1977] proposed a distributed routing algorithm which is essentially a variation of the reduced gradient method. Since then, this algorithm has been generalized by Segall [1979] for virtual circuit networks and by Ephremides [1978] for mixed media networks. However, Gallager's algrithm suffers from slow convergence and the difficulty in determining a proper stepsize from the input levels.

To curtail this problem, Bertsekas [1979], and Bertsekas, Gafni and Gallager [1984] use another class of gradient projection methods originally due to Goldstein [1964], and Levitin and Poljak [1966]. Since the constraints of the multicommodity flow problem are simple, the projections in this class can be computed simply. In addition, a more elaborate form of this class, the projected Newton method, is able to attain superlinear convergence rate, while keeping the computation overhead per iteration moderate (see Bertsekas [1982a]). However, the projected Newton method is not suitable for distributed implementation. Alternately the projected quasi-Newton routing algorithms by Bertsekas, Gafni and Gallager [1984] which attains a linear rate of convergence is suitable for distributed implementation. Corresponding to these algorithms are the related second derivative algorithms

(Bertsekas[1980], Bertsekas and Gafni [1983]) operating in the space of path flows. The path-flow algorithms have the advantage that loop-free properties of the paths do not need to be carefully guarded by the algorithm. Hence typically the implementation protocols and computations are simpler in form. The projected Newton method in path flow formulation by Bertsekas and Gafni [1983] requires global information, thus is not suitable for distributed implementation.

Another interesting algorithm is the extremal flow method due to Cantor and Gerla [1974]. This method has the disadvantage that each iteration requires solutions to a nonlinear programming problem and a shortest path problem, thus the computational overhead per iteration can be excessive. The algorithm is also not suitable for distributed implementation.

To summarize, for a virtual circuit network, the gradient projection algorithms scaled by the second derivatives (Bertsekas [1982d]) seem to be the best choice for quasi-static routing. They have a small computational overhead, while converging faster than the FD method, they converge only slightly slower than the projected Newton method, when starting far from an optimal routing. In the context of quasi-static routing, as the input levels are varying, and the measurements are inherently inaccurate, it is not important for the algorithm to achieve fast convergence in a neighborhood of an optimal routing. This fact makes the gradient projection method more attractive than the projected Newton method.

1.2.3 Virtual Circuit Path Assignment

There are very few works in the literature that deal with optimizing virtual circuit path assignments during an update interval.

The closest work we can find is that of Yum [1981]. He considers the problem of implementing a given *optimal* static routing for a datagram network in order to minimize the steady-state average delay. He shows that a deterministic sequence of routing decisions, which approximates a randomized routing scheme, performs better than the pure randomized scheme.

Other related problems are those local dynamic routing problems whose goal is to assign routes at an origin node for the arriving customers to traverse a simple queueing network, in order that the average delay or a related cost is minimized. Most of such studies assume an infinite horizon of time, and discounted or average costs. When using discrete-state queueing models, they appear to employ exclusively continuous-time Markov chains that can be uniformized (see Keilson [1979]), i.e. the chains that are equivalent to their discrete-time counterparts. These two features simplify the optimality condition and often lead to closed-form threshold type of schemes. Unfortunately, our problem presumes a finite horizon of time and due to the assumption of infinite number of servers, our continuous-time chains cannot be uniformized.

A summary of the related local dynamic routing results are in order. Hajek [1982] uses little more than the well-known inductive dynamic programming approach (see [Stidham and Prabhu 1974; p. 282]) to show that a threshold type of metering rule is optimal for controlling two interactive service stations. This result generalizes the similar results by Rosberg, Varaiya and Walrand [1982], who use uniformization and discrete-time dynamic programming, and by Foschini and Salz [1978], who use a diffusion model. By employing both deterministic flow equivalent and stochastic discrete queueing models, Sarachik [1982, 1984] similarly discovers that the threshold type of metering rules minimizes the aggregate or average delay for some local dynamic routing problems. Similar results are obtained by Ephremides, Varaiya and Walrand [1980], using backward induction on a continuous-time dynamic programming formulation.

Finally we mention some general results in the optimal control of queueing systems. Two survey papers by Stidham and Prabhu [1974] and Sobel [1974] provide an introductory background. Optimality conditions via dynamic programming for general jump processes are derived by Boel and Varaiya [1977]. Related optimality results using different methods have been reported by Rishel [1975] (using the minimum principle), Stone [1973] and Borkar [1984].

1.3 Scope Of The Thesis

This thesis report is organized as follows. We first review the Gafni-Bertsekas synchronous model and the Tsitsiklis-Bertsekas distributed asynchronous model in chapter 2. We also construct a combined model upon which the problem of optimal routing in a stochastic network is formulated.

In chapter 3 we present a simple extension of the Gafni-Bertsekas synchronous shortest path routing convergence result. We also study shortest path routing with length functions taking values from a discrete set. An example is presented in section 3.3 which shows that shortest path routing can be fooled by the discrete lengths and fail to converge to optimality.

The main result of the thesis–the convergence of synchronous gradient projection routing with randomized path assignment–is presented in chapter 4. We also discuss algorithmic variations and statistics of path flows resulting from randomized path assignment.

The main contribution is that the gradient projection method does not require the update interval to be small in order for the long-term deviation from optimality to be small. As for the shortest path method, an additional condition that the routing interval is small (as compared to the average VC holding time) is needed to ensure small long-term deviation from optimality.

Chapter 5 contains the convergence result of asynchronous gradient projection routing with randomized path assignment. It is shown that if the update intervals and the time between consecutive receptions of measurements are bounded, the asynchronous gradient projection routing has a similar convergence result as that of the synchronous counterpart.

In chapter 6 we study the path assignment problem in detail. We propose a simple threshold type of scheme which assigns paths according to the differences between actual and desired numbers of virtual circuits. We then present a convergence result for gradient projection routing with metered path assignment which is similar to the corresponding result for gradient projection routing with randomized path assignment.

In chapter 7 we present simulation results. Specifically we discuss simulation parameters, the test network, and the various experiments of the simulation. It appears that when virtual circuits turn over (i.e. arrive and leave) *quickly*, shortest path routing performs noticeably worse than gradient projection routing. However, when virtual circuits turn over *slowly*, there is no major difference between the performances of both types of routings.

Finally we conclude in chapter 8 by providing some directions for further research.

CHAPTER TWO
THE QUASI–STATIC ROUTING MODELS

2.1 Introduction

In this chapter we review the quasi-static routing models developed by Gafni and Bertsekas [1983b], and Tsitsiklis and Bertsekas [1986]. These two models are combined to form the stochastic asynchronous routing model which is used in this thesis.

The main contribution of the synchronous model by [GB] is that they define explicitly the *many small users* assumption. The extent of violation of the *many small users* assumption is quantified by a single parameter.

In the past, most quasi-static routing models simply assume that the load is constant. However, the Gafni-Bertsekas Model considers the network as a set of $M/M/\infty$ queues with each VC as a customer. The VCs are assumed to arrive at Poisson rates and stay on the network with exponential durations. Furthermore, the traffic rate of each VC of the same OD pair is assumed to be constant. In this manner, the load is varying slowly over time and eventually converges to a statistical steady-state. The $M/M/\infty$ queue assumption merely reflects the fact that flow control is not in effect and any VC is allowed to sign on. This quasi-static model is better than most of the existing ones. Note that the model is stochastic solely because of the stochastic nature of VC arrivals and departures.

The asynchronous model developed by [TB] is useful for its simple characterization of delays in control packets, inaccurate flow measurement, and flow transients. However, the model is deterministic, incapable of capturing the stochastic variations in practical data networks. We shall have further remarks on this model in section 2.3.

The model that we use is a direct combination of the two above mentioned models, retaining all the good features of both models.

2.2 The Gafni–Bertsekas Synchronous Quasi–Static Routing Model

Consider a network described by a directed graph $G = (V, L)$, where V is the set of nodes, L is the set of directed links. A set W of generic origin-destination (OD) pairs is given. For any OD pair w in W, which represents a class of VCs originating from node i and ending at node j, a set of paths P_w consisting of loop-free paths from node i to node j is assumed given.

We assume, for any OD pair w, a VC arrival rate $\overline{\lambda}_w/\varepsilon$ (in VC/time unit) with a communication rate $\overline{\gamma}_w\varepsilon$ (in data unit/VC/time unit) per VC so that the average communication rate for the OD pair is kept at $\overline{\lambda}_w\overline{\gamma}_w$. Note that the smaller the parameter ε, the larger the VC arrival rate and the smaller the traffic rate per VC. The limit $\varepsilon \to 0$ is referred to as the *many small users* assumption, and ε is considered as a measure of the extent of violation of the assumption. In terms of control, the smaller the ε the better the routing controllers' ability to fine-tune their control. This is because a path flow is the sum of traffic rates of the VCs assigned on the path. A controller can adjust a path flow only by changing the the number of VCs on the path. In the limit that ε goes to zero, we have a fluid approximation of data flows.

Each conversation (or VC) for an OD pair w is assumed to have exponentially distributed holding time with mean $1/\mu_w$. Each OD pair w is assumed to be able to accommodate all arriving VCs, and each VC is assigned according to some rule to a path p in P_w. Hence each OD pair evolves as an $M/M/\infty$ queue with active VCs as customers.

Let $x_p(t)$ denote the flow on the path $p \in P_w$ at time t, then

$$x_p(t) = \overline{\gamma}_w \varepsilon N_p(t) \geq 0 \qquad \text{(in data unit/time unit)},$$

where $N_p(t)$ is the number of active VCs assigned on path p, and $\overline{\gamma}_w$ is a constant.

Eventhough the real communication rate of a VC is random, the rate $\overline{\gamma}_w$ used in the above equation is obtained by averaging the real rates over a long period of time and over all VCs of the OD pair w so that the variance of $\overline{\gamma}_w$ is so small that $\overline{\gamma}_w$ can be considered as a deterministic quantity. For each OD pair $w \in W$, let $r_w(t)$ be the total communication rate (in data unit/time unit), then

$$r_w(t) = \sum_{p \in P_w} x_p(t).$$

At times $t = nT,\ n = 0, 1, 2, \ldots$, where $T > 0$ is a fixed updating interval, the total data flow $F_{ij}(t)$ on each link (i, j) is measured and is assumed to be exactly equal to

$$F_{ij}(t) = \sum_{w \in W} \sum_{p \in P_w \ (i,j) \in p} x_p(t).$$

In order for the above flow conservation equation to hold, the queue associated with each transmission line must have reached a statistical steady-state – i.e., the average data flow into the queue is equal to the average data flow out of the queue. Hence the conservation of flow assumption is implicitly taken. Another assumption which is needed here is that the control packets do not use up any bandwidth. This assumption is clearly not true in practice since over-flooding of control packets does deteriorate the routing performance (see [McQuillan et. al. 1980]).

All VCs that arrive in the interval $(nT, (n+1)T]$ are assigned to a path p according to a rule depending on $F(nT) = \{F_{ij}(nT) : \ (i, j) \in L\}$, the link flow vector at time nT.

A separable convex cost function, reflecting the congestion level of the network,

$$\overline{D}(F) = \sum_{(i,j) \in L} \overline{D}_{ij}(F_{ij}),$$

is assumed to be given. For each link $(i, j) \in L$, the link cost $\overline{D}_{ij}$ defined on $[0, \infty)$ is assumed to be non-negative real-valued, convex, increasing and subdifferentiable. We shall further assume that the link costs are in general continuously differentiable, the nondifferentiable case is an exception. Assume that, for all $(i,j) \in L$, there exists a positive constant L indepent of (i,j) such that the derivative

$$\overline{d}_{ij}(\cdot) = \frac{d\overline{D}_{ij}}{dF_{ij}}(\cdot)$$

satisfies

$$|\overline{d}_{ij}(F_{ij}) - \overline{d}_{ij}(\overline{F}_{ij})| \leq \mathrm{L}|F_{ij} - \overline{F}_{ij}| \quad \forall F_{ij}, \overline{F}_{ij},$$

i.e. $\overline{d}_{ij}(\cdot)$ is uniformly Lipschitz continuous. This assumption excludes using (Kleinrock [1964]) average queue size as the cost : for all $0 \leq F_{ij} \leq C_{ij}$,

$$\overline{D}_{ij}(F_{ij}) = \frac{F_{ij}}{C_{ij} - F_{ij}}, \tag{1}$$

where C_{ij} is the capacity of the link (i,j).

However we can use the following modified link costs:

$$\overline{D}_{ij}(F_{ij}) = \begin{cases} Q_{ij}(F_{ij}), & \text{if } Fij \leq \beta C_{ij}, \\ Q_{ij}(\beta C_{ij}) + Q'_{ij}(\beta C_{ij})(F_{ij} - \beta C_{ij}) & \text{otherwise,} \\ +\frac{1}{2}Q''_{ij}(\beta C_{ij})(F_{ij} - \beta C_{ij})^2, & \end{cases} \tag{2}$$

where

$$Q_{ij}(F_{ij}) = \frac{F_{ij}}{C_{ij} - F_{ij}},$$

where $0 < \beta < 1$ is a fixed threshold. We assume that the optimal long-term average routing (to be defined shortly) does not have $F^*_{ij} = C_{ij}$, for any (i,j). Thus as long as β is close enough to unity, the optimal routing of the original problem using (1) is the same as that of the modified problem using (2). Thus the Lipschitz continuity assumption is not a real restriction.

We also remark that the choice of the cost function is not critically important either theoretically or practically. Simulation results have shown that so long as the link cost is a strictly increasing function of link flow and approaches infinity as the flow goes to the capacity, the optimal routing does not vary much over different choices of link costs (see [Vastola 1979]).

We are interested in comparing the routing $\{\overline{D}(F(t)), F(t)\}$ with an optimal long-term average routing $\{D^*, F^*\}$ which is the solution of the following optimization problem:

$$\begin{aligned} \text{min.} \quad & \overline{D}(F) \\ \text{s.t.} \quad & F \in \mathrm{X}_F, \end{aligned}$$

where X_F is the set of link flows F satisfying

$$\begin{aligned} F_{ij} &= \sum_{w \in W} \sum_{p \in P_w \; (i,j) \in p} x_p && \forall (i,j) \in L, \\ \sum_{p \in P_w} x_p &= \frac{\overline{\lambda}_w \overline{\gamma}_w}{\mu_w} && \forall w \in W, \\ x_p &\geq 0 && \forall p \in P_w, w \in W. \end{aligned}$$

In words, X_F is the set of all possible average total link rates resulting from the long-term (or steady-state) average input rate:

$$\overline{r}_w = \frac{\overline{\lambda}_w \overline{\gamma}_w}{\mu_w} \qquad \forall w \in W.$$

One would think that the routing should minimize the cost $\overline{D}(t)$ at each time t. However, such a goal calls for solutions to difficult dynamic routing problems. Thus we are satisfied, in a quasi-static situation, with comparing the steady-state routing performances.

It should be noted that the synchronous model developed so far ignores data-link-control. The omission of data-link-control is to make the analytical model mathematically more tractable. With the knowledge of initial number $N_p(0)$ of active VCs on each path p, we can characterize the statistics of all the processes of subsequent interests in this synchronous routing model.

In order to simplify the analysis, we now introduce an alternative cost defined in terms of path flows. Let $< \cdot, \cdot >$ and $\| \cdot \|$ denote the usual Euclidean inner product and its induced norm. Let $P = \cup_{w \in W} P_w$ be the set of all paths, $x = \{x_p : p \in P\}$ be any path flow vector, and $F = \{F_{ij} : (i,j) \in L\}$ be any link flow vector. For any path flow x, we can define the corresponding link flow by

$$F_{ij} = \sum_{w \in W} \sum_{p \in P_w \ (i,j) \in p} x_p,$$

or in matrix form :

$$F = Hx,$$

$$F_{ij} = < H_{ij}, x >,$$

where H is a matrix whose $((i,j),p)$-th element is one, if $(i,j) \in p$, or zero otherwise. H_{ij} is the (i,j)-th row of H. We are thus led to the alternate cost function :

$$D(x) = \sum_{(i,j) \in L} D_{ij}(x),$$

where

$$D_{ij}(x) = \overline{D}_{ij}\big(< H_{ij}, x >\big).$$

Clearly, D_{ij} inherits the convexity and smoothness properties of $\overline{D}_{ij}$.

Before presenting the results, we introduce some notations and general rules about the notations in this report. For any variable $y(t)$, we denote $y(nT)$ by $y(n)$, e.g.,

$$x_p(n) = x_p(nT), \qquad \overline{D}(n) = \overline{D}\big(F(nT)\big).$$

The followings are further notations that are needed subsequently:

$$x_w = \{x_p \ : p \in P_w\},$$

$$d_w(n) = \frac{\partial D}{\partial x_w}(x(n)),$$

$$d_p(n) = \frac{\partial D}{\partial x_p}(x(n)),$$

$$X_w = \left\{ x_w \, : \sum_{p \in P_w} x_p = \bar{r}_w, x_p \geq 0, \forall p \in P_w \right\},$$

$$X = \left\{ x \, : \sum_{p \in P_w} x_p = \bar{r}_w, x_p \geq 0, \forall p \in P_w, \forall w \in W \right\},$$

$$\tilde{x}_w(n) = \frac{x_w(n)\bar{r}_w}{r_w(n)} \qquad \text{(normalized path flow vector)},$$

$$\tilde{F}(n) = H\tilde{x}(n) \qquad \text{(normalized link flow vector)},$$

$$M = \max\{\mu_w \, : w \in W\},$$

$$\mu = \min\{\mu_w \, : w \in W\},$$

and the system parameter set $\mathbf{A}$, which is the set of relevant parameters and initial conditions,

$$\mathbf{A} = \{\bar{r}_w, \bar{\lambda}_w, \mu_w, r_w(0) \, : \forall w \in W\} \cup \{T, \overline{D}, \delta, \Delta, G = (V, L)\},$$

where Δ, δ are the constants relating to scaling matrices of the gradient projection methods (see (1.2) of chapter 2).

Note that the normalized flow vectors, $\tilde{F}(n), \tilde{x}_w(n)$, are the flow vectors resulting from scaling the current load, $r_w(n)$, to the steady-state load, $\bar{r}_w$.

2.3 The Stochastic Asynchronous Model

The asynchronous model we use, based on the above quasi-static routing model, is obtained by substituting asynchronous updates for synchronous updates, and adding details about flow measurements and delays of control pockets.

First, we assume all routing updates, flow measurements, message transmissions and receptions occur at discrete time instants of the form nT, where $T > 0$ is the smallest time increment, and $n = 0, 1, 2, \ldots$. This assumption is not unrealistic, since a data network is a digital system and all relevant events can be ordered in terms of a global clock, but individual nodes do not need to have access to this global clock.

Flow Measurement

Suppose that a node i has the link (i,j) attached to it. Then at each time n, node i has an estimate of the instantaneous flow $F_{ij}(n)$:

$$\overline{F}_{ij}(n) = \sum_{m=n-Q}^{n} C_{ij}(n,m)F_{ij}(m),$$

where $C_{ij}(n,m)$ are non-negative (generally unknown) scalars summing to unity, and Q is a bound on the interval over which the measurements are averaged. This is a good representation of practical implementation- -most networks use some forms of averaging to estimate link flows.

Message Transmissions and Receptions

From time to time the local flow measurements are forwarded to the nodes which do not have the local measurements. This is usually done by a flooding mechanism, which involves some delay. At each node k, at time n, the controller receives or has available an estimate of $F_{ij}(n)$ for all $(i,j) \in L$:

$$\hat{F}_{ij,k}(n) = \overline{F}_{ij}(n - b_{ij,k}(n)),$$

for some $b_{ij,k}(n) \geq 0$. Let us assume that there exists $T_d > 0$ such that $b_{ij,k}(n) \leq T_d$, $\forall i,j,k,n$, then

$$\hat{F}_{ij,k}(n) = \sum_{m=n-G}^{n} d_{ij,k}(n,m)F_{ij}(m),$$

where $d_{ij,k}(n,m)$ are non-negative scalars summing to unity, and $G = Q + T_d$. Note that T_d is a bound on delays due to forwarding of control packets.

Asynchronous Updates

For each $w \in W$ we define, the set of update times,

$$T_w = \{n : \text{ a routing update is performed for } w \text{ at time } n\}.$$

These sets T_w are generally not known but we assume an upper bound between consecutive updates as follows. Assume that there is an $\overline{n} > 0$, such that $\forall\ w \in W, \forall n, m \in T_w, n, m$ are consecutive times in T_w,

$$|n - m| \leq \overline{n}.$$

In practice, routing updates are always carried out often enough just to keep the routing adapted to current loads. Hence the assumption is realistic.

In order to show the convergence result, we need a special definition. We define, for each $w \in W$, the latest update time before or at time n, $\overline{n}_w(n)$, to be

$$\overline{n}_w(n) = \max\{m \in T_w : m \leq n\}.$$

Hence

$$0 \leq n - \overline{n}_w(n) \leq \overline{n} \qquad w \in W, n = 0, 1, 2, \ldots$$

Desired (Target) Flows

The desired flows are only needed for gradient projection routing. They are referenced to as the desired flow rate on each path, between a routing update, for the purpose of monitoring actual flow rates.

At each routing update time n, the controller at the origin node of an OD pair w updates the desired flows for each path $p \in P_w$, $x_p^*(n+1)$. Thus if $n \notin T_w$, then $x_p^*(n+1) = x_p^*(n)$. In general, we have

$$x_w^*(n+1) = x_w^*(\overline{n}_w(n) + 1).$$

To calculate these desired flows, the controller needs to estimate the first derivative length (FDL) of paths, or the first derivatives of the cost against the path flows. Note that

$$\frac{\partial \overline{D}_{ij}}{\partial x_p}(x(n)) = \begin{cases} \overline{d}_{ij}(F_{ij}(n)) & \text{if } (i,j) \in p, \\ 0 & \text{otherwise.} \end{cases}$$

Hence, if k is the origin node for w, then $\forall n \in T_w, p \in P_w$, a natural estimate of the FDL of path p, $\eta_p(n)$, at n is

$$\eta_p(n) = \sum_{(i,j)\in P} \frac{d\overline{D}_{ij}}{dF_{ij}} \left(\hat{F}_{ij,k}(n)\right).$$

An alternative way is that the FDL for each link is estimated locally and forwarded to other nodes. This change will not affect the proof in any essential way, so we omit it. For convenience, we define, $\forall p \in P_w, \forall n \notin T_w$

$$\eta_p(n) = \eta_p(n-1),$$

that is, when no routing update is performed at n, it is not necessary to update the FDLs. In general, we have, $\forall n = 0, 1, 2, \ldots$

$$\eta_p(n) = \sum_{(i,j)\in P} \frac{d\overline{D}_{ij}}{dF_{ij}} \left(\hat{F}_{ij,k}(\overline{n}_w(n))\right),$$

where $\hat{F}_{ij,k}(\overline{n}_w(n))$ satisfies

$$\hat{F}_{ij,k}(\overline{n}_w(n)) = \sum_{m=n-C}^{n} g_{ij,k}(n,m) F_{ij}(m),$$

and $g_{ij,k}(n,m)$ are non-negative scalars summing to unity and $C = Q + T_d + \overline{n}$. Thus C is an upper bound on measurement intervals plus delays in control packets, and the bound on the time between consecutive routing updates.

The description of the asynchronous model is now complete. We also note that the system parameter set $\mathbf{A}$ should be enlarged to include the parameters, C, and $\overline{n}$.

CHAPTER THREE
SYNCHRONOUS SHORTEST PATH ROUTING

3.1 Introduction

Almost all the existing data networks use some forms of shortest path routing. This popularity deserves a thorough theoretical investigation.

In this chapter we first review the Gafni-Bertsekas [GB] convergence result and extend the result somewhat. Their convergence result on the link flows requires that for each link, the length function has a minimum slope. This condition may be too strong for some choices of cost functions. We have weakened this condition to strict convexity of the link costs.

We then consider shortest path routing with integer-valued length functions. This case is of interest because routing in some existing networks, e.g., TYMNET, takes the length values from a discrete set. We shall show that, by means of a counter-example, such discontinous length function can fool a shortest path routing algorithm and cause the routing to oscillate around a non-optimal point. This result further illustrates the *unstable* characteristics of shortest path routing.

3.2 Review Of The Gafni–Bertsekas Convergence Result

We review the Gafni-Bertsekas convergence result on synchronous shortest path routing here because this result will be compared with the counterpart on gradient projection routing later in this report. In addition we present an extension of their result.

The result, Theorem 3.1, basically states that, on the average, the total network cost decreases at an exponential rate to a neighborhood of the steady-state minimum cost. The rate of cost decline is lower bounded by the slowest VC departure rate in the network.

In addition, the smaller the update interval and the extent of violation of the *many small users* assumption, the smaller the long-term deviation from optimality.

This result is important in the sense that it ensures that VC shortest path routings do converge to a neighborhood of an optimal performance. The main reason that the result holds is that if VCs are not re-routed, only the arriving VCs are routed via the current shortest paths, then the routing is behaving like a stochastic Frank-Wolfe algorithm. As the deterministic Frank-Wolfe algorithm for the multi-commodity flow problem has been shown to converge, the stochastic version should likewise converge. Unlike the deterministic case, the shortest path routing does not vary its stepsize–there is no stepsize to choose, only the routing update interval to adjust. Together with the fact that path flows cannot be infinitesimally divided, the convergence is only to a neighborhood. For practical purposes, if the neighborhood is small, a routing close to the optimum is acceptable. Since the load is always varying, it does not make sense to match the long-term optimal routing exactly.

The last part of the Gafni-Bertsekas result is that if the length functions have some minimum slopes then the link flow vector converges to the unique optimal flow vector, as both the extent of violation of the *many small users* assumption and the update interval tend to zero. However, such a condition on length functions clearly implies the strict convexity of the cost. It seems possible to weaken this condition to just strict convexity– since if the cost is strictly convex, the optimal flow vector must be unique. In fact we prove this conjecture in Theorem 3.1.

The shortest path routing algorithm is defined as follows. All VCs of an OD pair w arriving during the interval $(nT, (n+1)T]$ are assigned to a path $\overline{p}_w \in P_w$ which is the shortest according to the first derivative lenths (FDL), i.e.,

$$d_{\overline{p}_w}(n) = \min\{d_p(n) \ : p \in P_w\},$$

where

$$d_p(n) = \sum_{(i,j)\in p} \overline{d}_{ij}(n).$$

We assume a fixed deterministic rule to resolve ties between paths. Let $F(n)$ (or $x(n)$) denote the link (or path) flow vector corresponding to the above shortest path routing.

Theorem 3.1. (a) There exist positive constants, c_1, c_2, $a(\varepsilon,T)$, $b(\varepsilon,T)$, which depend only on the system parameter set $\mathbf{A}$, such that $\forall n = 0,1,2,\ldots$

$$-c_1 e^{-\mu n T} \leq E\left[D(n) - D^*\right] \leq e^{-\mu n T}\left[D(0) - D^*\right] + c_2\left[a(\varepsilon,T) + b(\varepsilon,T) n T e^{-\mu n T}\right],$$

and that

$$\lim_{\varepsilon\to 0}\lim_{T\to 0} a(\varepsilon,T) = 0, \qquad \lim_{\varepsilon\to 0}\lim_{T\to 0} b(\varepsilon,T) < \infty,$$

$$\lim_{\varepsilon\to 0}\lim_{T\to 0}\overline{\lim_{n\to\infty}} E[\overline{D}(n)] = D^*.$$

(b) If, in addition, $\overline{D}_{ij}$ is strictly convex for all $(i,j)\in L$, then

$$\lim_{\varepsilon\to 0}\lim_{T\to 0}\overline{\lim_{n\to\infty}} E\left(\|F(n) - F^*\|^2\right) = 0.$$

Proof. (a) is proved in [GB].

To show (b) we use the convexity result:

$$\begin{aligned}\overline{D}(\tilde{F}(n)) \leq& \overline{D}(F(n)) + \left\langle \frac{dD}{dF}(\tilde{F}(n)), \tilde{F}(n) - F(n)\right\rangle \\ \leq& \overline{D}(F(n)) + A_1\|F(n) - \tilde{F}(n)\|,\end{aligned}$$

where A_1 is a positive constant, and the second inequality comes from the fact that $\tilde{F}(n) \in \mathrm{X}_F$, and X_F is a compact set. We have, using (a) and (A13) of [GB],

$$\begin{aligned}\lim_{\varepsilon\to 0}\lim_{T\to 0}\overline{\lim_{n\to\infty}} E[\overline{D}(\tilde{F}(n)] \leq& \lim_{\varepsilon\to 0}\lim_{T\to 0}\overline{\lim_{n\to\infty}}\left\{E[\overline{D}(F(n))] + A_1 E\|\tilde{F}(n) - F(n)\|\right\} \\ =& D^*. \end{aligned} \qquad (1)$$

Since $\tilde{F}(n)$ is in the compact set X_F, and D^* is the minimal cost over X_F, we have, using the Jensen Inequality, $\forall n = 0, 1, 2, \ldots$

$$0 \le \overline{D}\left[E(\tilde{F}(n))\right] - D^* \le E\left[\overline{D}(\tilde{F}(n))\right] - D^*.$$

By (1)

$$\lim_{\varepsilon \to 0} \lim_{T \to 0} \overline{\lim_{n \to \infty}} \overline{D}\left[E(\tilde{F}(n))\right] = D^*.$$

This fact with the strict convexity of $\overline{D}$ imply

$$\lim_{\varepsilon \to 0} \lim_{T \to 0} \overline{\lim_{n \to \infty}} E[\tilde{F}(n)] = F^*.$$

Now by the Cauchy-Schwarz Inequality

$$E\|F(n) - F^*\|^2 \le 3E\|F(n) - E(F(n))\|^2 + 3\|E(F(n)) - E(\tilde{F}(n))\|^2 + 3\|E(\tilde{F}(n)) - F^*\|^2.$$

Note that, by (A11) of [GB], there exists a positive constant A_2 such that

$$E\|F(n) - E(F(n))\|^2 \le \|H\|^2 \sum_{p \in P_w} \text{Var}(x_p(n)) \le A_2 \varepsilon.$$

Also with Jensen Inequality and (A13) of [GB]

$$\begin{aligned} \|E(F(n)) - E(\tilde{F}(n))\|^2 \le& \|H\|^2 \|E[x(n)] - E[\tilde{x}(n)]\|^2 \\ \le& \|H\|^2 \sum_{p \in P_w} E\left[|x_p(n) - \tilde{x}_p(n)|^2\right] \\ \le& A_3 \varepsilon + A_4 e^{-\mu n T}, \end{aligned}$$

where A_3 and A_4 are some positive constants. The above two inequalities now imply

$$\begin{aligned} \lim_{\varepsilon \to 0} \lim_{T \to 0} \overline{\lim_{n \to \infty}} E\|F(n) - F^*\|^2 \le& \lim_{\varepsilon \to 0} \lim_{T \to 0} \overline{\lim_{n \to \infty}} \{(A_2 + A_3)\varepsilon + A_4 e^{-\mu n T} \\ &+ \|E(\tilde{F}(n)) - F^*\|^2\} \\ =& \|\lim_{\varepsilon \to 0} \lim_{T \to 0} \overline{\lim_{n \to \infty}} E(\tilde{F}(n)) - F^*\|^2 = 0, \end{aligned}$$

which proves the assertion (b). *Q.E.D.*

Some additional comments on shortest path routing are in order. The *unstable* behavior of shortest path routing is well-known and the interested reader is referred to

[Bertsekas and Gallager 1986] and [Bertsekas 1982c] for detailed treatment. One of the causes of this instability is that the average VC holding time is very short as compared to the update interval. Consider the extreme case in which, during an update interval, all the VCs which arrived before the latest update depart before the next update, then all the new VCs arriving during the interval are routed via the shortest paths. This results in overloading the shortest paths. This phenomenon repeats itself during the next update interval. This is a reason why the update intervals in shortest path routing should be small as compared to the average VC holding time. In fact, according to Theorem 3.1, this condition is needed to ensure the convergence of the routing.

3.3 Shortest Path Routings with Integer-Valued Link Length

The routing problem considered in this section is quite different from the earlier one–the first derivative link lengths are integer-valued and the VC arrival and departure processes are deterministic. The assumption that the length functions are integer-valued merely reflects that the length functions only take values from a discret set. With the length functions being discontinuous, shortest path routing may not converge to optimality. We shall first derive an optimality condition for the routing problem, then construct a *deterministic* example showing the non-convergence of shortest path routing. The example in fact shows that, under certain initial conditions, the routing may oscillate around *seemingly* optimal but *actually* non-optimal points, and be incapable of finding a descent step. Since the example is deterministic, we do not know whether shortest path routing will converge or not in a stochastic environment.

First we need to introduce some additional terminology related to the discontinuity in the length functions. A link (or path) flow vector F(or x) is said to be a break point if a link (or path) length is discontinuous at F(or x) . Let F_{ij} be a discontinuity point of a link length $\overline{d}_{ij}(\cdot)$, one can define $\overline{d}_{ij}(F_{ij})$ to be any value between the two integers

$$\lim_{c>0\ c\downarrow 0} \overline{d}_{ij}(F_{ij} + c) \qquad \text{and} \qquad \lim_{c>0\ c\downarrow 0} \overline{d}_{ij}(F_{ij} - c).$$

We shall call these values admissible values. Correspondingly, at a continuity point of a length $\overline{d}_{ij}$, the length is well-defined and is also referred to as an admissible value. Without loss of generality, we shall assume that each link length has finitely many different values. Then by standard results of convex analysis (e.g., Rockafellar [1970]), D and $\overline{D}$ are convex polyhedral functions, and the subdifferential $\partial\overline{D}$ at F is the set of link length vectors whose coordinates take admissible values. The subdifferential ∂D at x is the set of path length vectors definable using link length vectors in $\partial\overline{D}(Hx)$. For each path flow vector x, any subgradient v in $\partial\overline{D}(Hx)$ is said to be an admissible path length for x. The following figure provides an example of a break point and its subdifferential.

$$\overline{D}(F) = \int_0^F d_{12}(F)dF \qquad \partial\overline{D}(1) = [1,2]$$

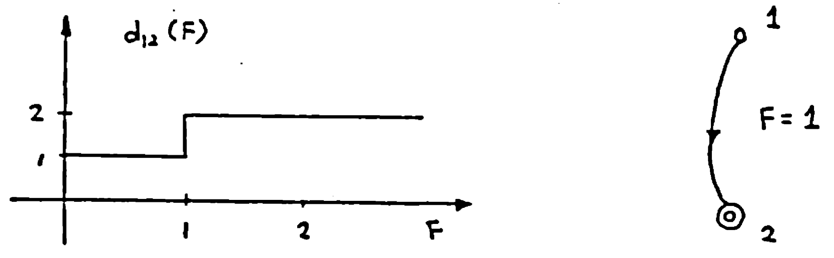


Figure 3.1 An Example of Subgradients

From the above example, the set of admissible lengths is the convex hull of the two extremal lengths, 1 and 2. The next lemma provides the optimality condition for routing.

Lemma 3.2 A path flow x in X is optimal for the deterministic optimal routing problem, i.e. $D(x) = D^*$, if and only if there exists an admissible path length vector v for

x, such that for each OD pair w, x has positive flows only on shortest paths according to path length vector v. In other words, x is optimal if and only if $\forall p \in P_w, x_p > 0$ implies $v_p = \min\{v_{\overline{p}} : \forall \overline{p} \in P_w\}$.

Proof By Theorem 27.4 of Rockafellar[1970], x is optimal

$$\iff \exists v \in \partial D(x), \quad \text{s.t.} \quad \sum_{p \in P} v_p(y_p - x_p) \geq 0 \; \forall y \in X$$

$$\iff \exists v \in \partial D(x), \quad \text{s.t.} \quad x_p > 0 \Longrightarrow \; v_p \leq v_{\overline{p}} \; \forall \overline{p} \in P_w \; \forall w \in W.$$

Q.E.D.

The following is the counter-example.

Example 3.3 Consider the following network where the origin and the destination nodes are connected by three directed links a_1, a_2, a_3; the links carry flows F_1, F_2, and F_3 respectively. Assume two OD pairs w_1 and w_2 and the following:

$$r_{w_1} = 3 \qquad r_{w_2} = 3,$$

$$p_{w_1} = \{p_1, p_2\} \qquad p_1 = a_1,\; p_2 = a_2,$$

$$p_{w_2} = \{p_3, p_4\} \qquad p_3 = a_2,\; p_4 = a_3.$$

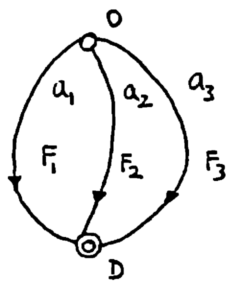


Figure 3.2 The Graph of the Counter-Example Network

Hence

$$F_1 = x_1,$$

$$F_2 = x_2 + x_3,$$

$$F_3 = F_4.$$

The overall cost function is given by

$$\overline{D}(F) = \int_0^{F_1} d_1(F)dF + \int_0^{F_2} d_2(F)dF + \int_0^{F_3} d_3(F)dF,$$

where d_1, d_2, d_3, and the initial flows are specified below :

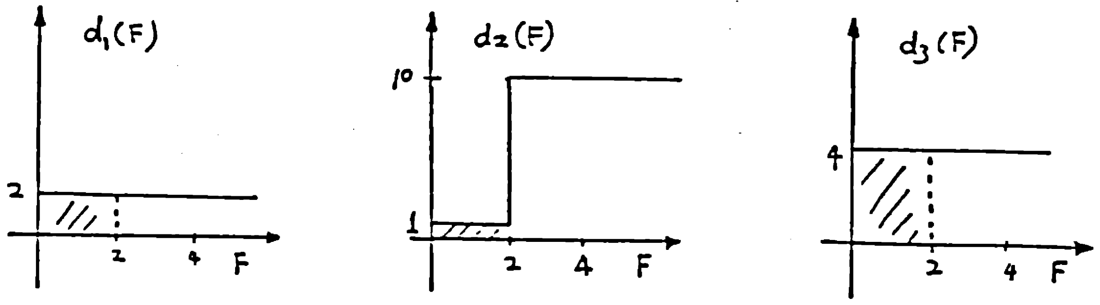


Figure 3.3 The Initial Loads

By Lemma 3.1, it is easy to check that the unique optimal routing is

$$x_1 = 3, \qquad x_2 = 0,$$

$$x_3 = 2, \qquad x_4 = 1.$$

In the limit, $\varepsilon \to 0$, the updating equation for shortest path routing is given (using e.g., Lemma A1 of [GB]) by the following: at iteration n, for each OD pair w, if p_j is the shortest first-derivative length path for w, then the average flows are updated by

$$x_i(n+1) = x_i(n)e^{-\mu T},$$

$$x_j(n+1) = 3 - x_i(n+1),$$

where p_j is the other path in P_w.

If we implement the iterations by assuming $d_2(2) = 1$, we get

$$x_1(0) = 2., \quad x_1(1) = 2e^{-\mu T}, \quad x_1(2) = 1 + e^{-\mu T} + 2(1 - e^{-\mu T})^2,$$

$$x_2(0) = 1., \quad x_2(1) = 1 + 2(1 - e^{-\mu T}), \quad x_2(2) = e^{-\mu T} + 2e^{-\mu T}(1 - e^{-\mu T}),$$

$$x_3(0) = 1., \quad x_3(1) = 1 + 2(1 - e^{-\mu T}), \quad x_3(2) = e^{-\mu T} + 2e^{-\mu T}(1 - e^{-\mu T}),$$

$$x_4(0) = 2., \quad x_4(1) = 2e^{-\mu T}, \quad x_4(2) = 1 + e^{-\mu T} + 2(1 - e^{-\mu T})^2.$$

Thus the routing is oscillating around the break point $x = \{2, 1, 1, 2\}$. Indeed by the summetry between the two OD pairs, $x_1(n) = x_4(n)$, $x_2(n) = x_3(n)$ for all $n = 0, 1, 2, \ldots$, and the algorithm does not converge to the unique optimal routing. A similar situation results if we assume $d_2(2) = 10$. One can also construct a similar example with $x(0)$ in a neighborhood of the break point $\{2, 1, 1, 2\}$, and still obtain non-convergence.

Observe that link a_2 is a bottleneck link shared by the two paths p_2 and p_3, and the marginal decrease in cost per unit decrease in x_2 is 2, while the marginal decrease in cost per unit decrease in x_3 is 4. Thus if we exchange one unit of flow x_3 for one unit of flow x_2 in link a_2, the overall cost decreases by 2. Thus a good routing algorithm should implement this kind of beneficial trading if it recognized a differential in marginal costs of the paths (of different OD pairs) sharing a bottleneck link. The problem here is that the algorithm does not coordinate the OD pairs to do such kind of trading. Note that when the marginal cost $d_2(2)$ is defined to be either the right-hand or the left-hand limit, such benefit is not obvious from the lengths.

Suppose $d_2(2)$ is defined to be any value in the open interval (2,4), then the first iteration ($n = 1$) will be a descent step. This observation suggests that a good routing should be able to dynamically adjust link lengths at or around break points (the interval (2,4) here actually depends on time-varying path lengths).

Note that for both OD pairs, the routing is oscillating around a point which seems optimal to them individually (applying Lemma 3.2 only to each OD pair respectively). Thus we have a situation where individually optimal solutions do not constitute a socially optimal solution.

CHAPTER FOUR
SYNCHRONOUS GRADIENT PROJECTION ROUTING WITH RANDOMIZED PATH ASSIGNMENT

4.1 Introduction

In this chapter, we prove the convergence of two classes of gradient projection algorithms implemented by randomizing path assignments at origin nodes. The result contained in this chapter is the core of this thesis.

There is a main advantage of the gradient projection algorithm that can be gleaned from the analysis. That is, if the load is not changing, the algorithm decreases the average total cost from a routing update to another, independently of the length of the update interval. For the shortest path routing algorithm, a decrease in average cost can be assured if the update interval is small.

One reason for this property is that the gradient projection algorithm has a stepsize at its disposal. As long as the stepsize is well chosen, the algorithm will not shift large amounts of flow to the shortest paths, thus avoiding the problem of overloading the shortest paths associated with the shortest path algorithm. Intuitively, the gradient projection algorithm does not over-react to good news (shortest paths) as well as bad news (long paths). The algorithm will shift flows to shortest paths gradually over a number of updates. This kind of conservative strategy will tend to reduce oscillations of the routing.

The convergence result is similar to that of shortest path routing, except that the only sufficient condition needed for the long-term deviation from optimality to go to zero is that the extent of violation of the *many small users* assumption tends to zero. This result is important in practice, since the smaller the updating frequency the fewer the communication resources which are used for forwarding update packets.

We chose to analyze the routing implemented by randomizing path assignments at origin nodes. The main reason is that this choice greatly simplifies the mathematics and still provides an desired convergence result. As mentioned earlier, a deterministic path assignment strategy produces flow statistics that are very difficult to calculate analytically. We shall turn to that problem in chapter 6.

We first introduce different versions of the gradient projection routing algorithm in section 4.2. The statistics of path flows which are needed in deriving and understanding the result are presented in section 4.3. Some lemmas which are needed in the main proof are reported in section 4.4. Two of these lemmas in fact show the descent property of the gradient projection algorithm. Finally the convergence result is presented in section 4.5 along with some interpretations and its proof.

4.2 Algorithmic Variations

There are some criteria that a good routing algorithm should satisfy. First, the routing variables, in our case, the desired flows, must be easily computable. Second, the routing must adapt to the current load since load variations affect the network congestion. Third, the desired flows should not be dependent on VC arrival and departure rates. In practice, these rates may not be Poisson and actually vary with time. A robust routing algorithm should not depend on the estimates of these rates.

We choose two versions of gradient projection algorithms for our analysis. Both are adaptive in the sense that they both use the current path flows to compute the next desired flows at an update. The first version is based on projection onto the simplex constraints and is computationally more demanding than the second one. The second version is based on projection onto the orthant constraints. Its computational overhead is almost equal to that of the shortest path algorithm. We also briefly mention another version in which the next desired flow is based on the current desired flows. This version is not so *adaptive*–it is included for completeness purposes.

We assume that at time nT for each $w \in W$, the desired flow $x_w^*(n+1)$ for the next interval $(nT, (n+1)T]$ is computed at the origin node of w, using the equation

$$x_w^*(n+1) = \left[x_w(n) - \alpha M_w^{-1}(n) d_w(n)\right]_{M_w(n)}^+ . \tag{1.1}$$

Here α is a positive scalar stepsize and $M_w(n)$ is a symmetric positive definite scaling matrix, $[\cdot]_{M_w(n)}^+$ denotes the projection onto the simplex

$$X_w(n) = \left\{ x_w \ : \sum_{p \in P_w} x_p = r_w(n), x_p \geq 0, \forall p \in P_w \right\}$$

with respect to the norm $||\cdot||_{M_w(n)}$ induced by $M_w(n)$. $M_w(n)$ is typically an approximation of the Hessian matrix $\partial^2 D / \partial x_w^2(n)$ (for the purpose of increasing the convergence rate); with such a choice the iteration becomes a projected quasi-Newton update. Since our analysis does not concern with increasing the convergence rate, we shall not specify further about $M_w(n)$. However, the following uniform bounds are assumed:

$$0 < \delta I \leq M_w(n) \leq \Delta I \quad \forall n, w, \tag{1.2}$$

where δ and Δ are positive constants and I is the identity matrix of appropriate dimensions.

In practice, the projection in the above algorithms can be computationally cumbersome. It requires the solution of a quadratic programming problem, for solving the projection, at each update. However, if the scaling matrices $M_w(n)$ are diagonal, the computational burden is only moderate. On the other hand, the second version which we now introduce uses the orthant constraints onto which the projections are simply computable.

For each $w \in W$, at each updating epoch nT, the desired flow $x_w^*(n+1)$ is computed as follows. Let $\overline{p}_w$ be a shortest path in P_w with respect to FDL $d_p(n)$, i.e.,

$$d_{\overline{p}_w}(n) = \min\{d_p(n) \ : p \in P_w\}.$$

Let

$$\hat{x}_w(n) = \{x_p(n) \ : p \in P_w, p \neq \overline{p}_w\}, \tag{2.1}$$

$$\hat{d}_w(n) = \{d_p(n) - d_{\overline{p}_w}(n) \ : p \in P_w, p \neq \overline{p}_w\}. \tag{2.2}$$

Assume that at time nT, a set of diagonal positive definite matrices $\hat{M}_w(n)$ is given $\forall w \in W$,

$$\hat{M}_w(n) = \operatorname{diag}(\hat{m}_{w,p}(n)).$$

These scaling matrices also satisfy the following condition: $\forall w \in W, n = 0, 1, 2, \ldots$

$$0 < \delta I \leq \hat{M}_w(n) \leq \Delta I,$$

where I is the identity matrix of suitable size. Then

$$\hat{x}^*_w(n+1) = \left[\hat{x}_w(n) - \alpha \hat{M}_w^{-1}(n)\hat{d}_w(n)\right]^+, \tag{3.1}$$

$$x^*_{\overline{p}_w}(n+1) = r_w(n) - \sum_{p \in P_w \ p \neq \overline{p}_w} x^*_p(n+1), \tag{3.2}$$

where $[\cdot]^+$ is defined by:

$$i - \text{th element of } [x]^+ = \max(0, x_i).$$

The third version is adapted from [TB]:

$$\begin{aligned} \hat{x}^*_w(n+1) &= \left[\hat{x}^*_w(n) - \alpha \hat{M}_w^{-1}(n)\hat{d}_w(n)\right]^+, \\ x^*_{\overline{p}_w}(n+1) &= r_w(n) - \sum_{p \in P_w \ p \neq \overline{p}_w} x^*_p(n+1). \end{aligned} \tag{4}$$

The only place (4) differs from (3) is that in (4) the next desired flows are computed from the current desired flows. This version is not as adaptive as the earlier versions in the sense that the current flows are not used in computing the next desired flows, only the current load $r_w(n)$ is used. We conjecture that a similar convergence result should be obtainable.

4.3 The Statistics of Path Flows

The key statistics needed for the subsequent proofs are the mean and variance of transient path flows resulting from an update. Following [GB], we assume that each OD w evolves like an $M/M/\infty$ queue with exponential servers at μ_w service rate.

Consider an OD pair w. Let $x_w^*(n+1)$ be the desired (target) flow computed at time nT. During the interval $(nT, (n+1)T]$, all arriving VCs are assigned to path $p \in P_w$ with fixed probability $u_p(n)$:

$$u_p(n) = \frac{x_p^*(n+1)}{r_w(n)}.$$

Define, at time nT, the desired number $N_p^*(n)$ of VCs on path p by

$$N_p^*(n) = \frac{x_p^*(n+1)}{\varepsilon \overline{\gamma}_w} \qquad \forall p \in P_w, \tag{5.1}$$

and the total number $n_w(n)$ of VCs by

$$N_w(n) = \sum_{p \in P_w} N_p^*(n) = \frac{r_w(n)}{\varepsilon \overline{\gamma}_w}. \tag{5.2}$$

Then the probability $u_p(n)$ is in fact a ratio of the desired number of VCs to the total number of VCs :

$$u_p(n) = \frac{N_p^*(n)}{N_w(n)}.$$

We shall also need the normalized flow $\tilde{x}_w^*(n+1)$ defined by

$$\tilde{x}_w^*(n+1) = \frac{x_w^*(n+1)\overline{r}_w}{r_w(n)}. \tag{6}$$

Let $x(n)$(or $F(n)$) be the path (or link) flow at time nT resulting from the gradient projection algorithm with the above randomized path assignment scheme.

Lemma 4.1 For all $n = 0, 1, 2, \ldots$ and $w \in W$

$$E[r_w(n)] = \overline{r}_w + e^{-\mu_w nT}[r_w(0) - \overline{r}_w], \tag{7.1}$$

$$\mathrm{Var}[r_w(n)] = \varepsilon \overline{\gamma}_w \left(1 - e^{-\mu_w nT}\right) \left[\overline{r}_w + e^{-\mu_w nT} r_w(0)\right]. \tag{7.2}$$

Furthermore, for each $w \in W$, each $p \in P_w$,

$$E[x_p(n+1)|x(n)] = \tilde{x}_p^*(n+1) + e^{-\mu_w T}[x_p(n) - \tilde{x}_p^*(n+1)], \tag{8.1}$$

$$\text{Var}[x_p(n+1)|x(n)] = \varepsilon\overline{\gamma}_w\left(1 - e^{-\mu_w T}\right)\left[\tilde{x}_p^*(n+1) + e^{-\mu_w T}x_p(n)\right]. \tag{8.2}$$

Proof. The first half of the result follows directly from Lemma A1 of [GB]. Now each path can be represented by an $M/M/\infty$ queue with arrival rate $u_p(n)\overline{\lambda}_w/\varepsilon$ and service rate μ_w, by equation (A8) of [GB]:

$$E[N_p(n+1)|x(n)] = \frac{\overline{\lambda}_w n_p^*(n)}{\varepsilon\mu_w n_w(n)} + e^{-\mu_w T}\left[N_p(n) - \frac{\overline{\lambda}_w n_p^*(n)}{\varepsilon\mu_w n_w(n)}\right].$$

Multiplying both sides by $\varepsilon\overline{\gamma}_w$, it yields

$$E[x_p(n+1)|x(n)] = \frac{\overline{r}_w n_p^*(n)}{n_w(n)} + e^{-\mu_w T}\left[x_p(n) - \frac{n_p^*(n)\overline{r}_w}{n_w(n)}\right],$$

which is equivalent to (8.1). Similarly by equation (A9) of [GB], we have

$$\text{Var}[N_p(n+1)|x(n)] = \left(1 - e^{-\mu_w T}\right)\left[\frac{\overline{\lambda}_w n_p^*(n)}{\varepsilon\mu_w n_w(n)} + e^{-\mu_w T}N_p(n)\right].$$

Multiplying both sides by $\varepsilon^2\overline{\gamma}_w^2$, it yields (8.2). *Q.E.D.*

The implication of the above lemma is that if the updating period is long as compared to the VC time constant $1/\mu_w$ (i.e. $T \to \infty$), the average flows will be close to the long-term average (normalized) desired flows. Furthermore, if the extent of violation of *many small users* assumption converges to zero (i.e. $\varepsilon \to 0$), the flows converge in mean square to the average desired flows exponentially at the natural rate μ_w.

Alternately, the smaller the ε the smaller the variance of the flows. In the limit $\varepsilon \to 0$, the path flows become essentially deterministic. The results (7) and (8) are in fact one of the keys to show the convergence result Theorem 4.5. Also the results (7) and (8) imply that the mean flows converge to the long-term averages.

We have seen that the convergence rate of the shortest path algorithm critically depends on μ (see Theorem 3.1). Without re-routing, there is no way for the routing controllers to shift flows faster than the slowest VC departure rate. Hence the convergence of the gradient projection algorithm cannot be much faster than the smallest μ_w, i.e. μ.

We need to introduce another notation, the step at time n:

$$s(n) = x^*(n+1) - x(n), \tag{9}$$

which is the difference between the desired flow and the current flow at nT. Note that when the routing algorithm converges to optimality, the desired flows should equal to the current flows, i.e., $s(n) = 0$. Hence, an important part of the convergence proof is to show that the step converges to zero in some sense. The next lemma gives a bound on the second order terms in the Taylor series expansion of D.

Lemma 4.2 There exist positive constants $A_1 - A_6$ depending only on system parameter set $\mathbf{A}$, such that $\forall n = 0, 1, 2, \ldots$

$$E\left[\|x(n+1) - x(n)\|^2 | x(n)\right] \leq A_1 \varepsilon \sum_{p \in P} \left[\tilde{x}_p^*(n+1) + e^{-\mu T} x_p(n)\right]$$
$$+ A_2 \left(\|s(n)\|^2 + \|\tilde{x}^*(n+1) - x^*(n+1)\|^2\right), \tag{10.1}$$

and for all $p \in P$

$$E\left[|\tilde{x}_p^*(n+1) - x_p^*(n+1)|^2\right] \leq A_3 \varepsilon + A_4 e^{-\mu n T}, \tag{10.2}$$
$$E\left[\|x(n+1) - x(n)\|^2\right] \leq A_5 \varepsilon + A_2 E\left[\|s(n)\|^2\right] + A_6 e^{-\mu n T}. \tag{10.3}$$

Proof. Consider any $w \in W$, for any $p \in P_w$, $n = 0, 1, 2, \ldots$, using (8), we get

$$E\left[|x_p(n+1) - x_p(n)^2 | x(n)\right]$$
$$= \mathrm{Var}\left[x_p(n+1) | x(n)\right] + \left[x_p(n) - E\left\{x_p(n+1) | x(n)\right\}\right]^2$$

$$
\begin{aligned}
&= \epsilon \overline{\gamma}_w \left(1 - e^{-\mu_w T}\right) \left[\tilde{x}_p^*(n+1) + e^{-\mu_w T} x_p(n)\right] \\
&\quad + \left(1 - e^{-\mu_w T}\right)^2 \left(\tilde{x}_p^*(n+1) - x_p(n)\right)^2 \\
&\leq \epsilon \overline{\gamma}_w \left(1 - e^{-\mu_w T}\right) \left[\tilde{x}_p^*(n+1) + e^{-\mu_w T} x_p(n)\right] \\
&\quad + 2\left(1 - e^{-\mu_w T}\right)^2 \left(|s_p(n)|^2 + |\tilde{x}_p^*(n+1) - x_p^*(n+1)|^2\right),
\end{aligned}
\tag{11}
$$

where the inequality comes from (9). Summing (11) over all $p \in P$, we get (10.1). By (10), the following inequalities hold for any w and any $p \in P_w$:

$$
\begin{aligned}
E\left[\sum_{p \in P_w} \tilde{x}_p^*(n+1)\right] &= \overline{r}_w, \\
E\left[\sum_{p \in P_w} x_p(n)\right] &= E[r_w(n)] \leq \overline{r}_w + |r_w(0) - \overline{r}_w|, \\
E\left[|\tilde{x}_p^*(n+1) - x_p^*(n+1)|^2\right] &= E\left[\left|\tilde{x}_p^*(n+1)\left(1 - \frac{r_w(n)}{\overline{r}_w}\right)\right|^2\right] \\
&\leq A_3 \epsilon + A_4 e^{-\mu n T},
\end{aligned}
$$

which is (10.2). Using the above inequalities and taking expectation on both sides of (10.1), we get (10.3). *Q.E.D.*

4.4 Preliminary Lemmas

Several lemmas which are needed to show the convergence result are presented here. Lemmas 4.3 and 4.4 are for the algorithm with the simplex constraints, whereas lemmas 4.3.a and 4.4.a are the counterparts for the algorithm with the orthant constraints.

Lemmas 4.3 and 4.3.a are important in the sense that they show that a gradient projection update is a descent step. More precisely, the lemmas show that if the flow at $n+1$ is precisely the desired flow for $n+1$, then the routing decreases a linearized cost from n to $n+1$ (the linearized cost is the first order Taylor series expansion of the cost at time n). In the following lemmas, recall that α is the stepsize, δ and Δ are respectively the smallest and the largest eigen values of the scaling matrices.

Lemma 4.3 For all $w \in W$ and for all $n = 0, 1, 2, \ldots$

$$\langle d_w(n), s_w(n) \rangle \leq -\frac{\delta}{\alpha} \|s_w(n)\|^2. \tag{12}$$

Proof. Let M be a symmetric positive definite matrix. We define the inner product $< \cdot, \cdot >_M$ and the associated norm $\| \cdot \|_M$ by

$$< x, y >_M = < x, M\, y >,$$

$$\|x\|_M^2 = < x, M\, x > .$$

In $R^{|w|}$, let $[d]_M^+$ be the projection of d onto a closed convex subset G with respect to the norm $\| \cdot \|_M$. By the definition of the projection $[\cdot]_M^+$, we have

$$\left\langle d - [d]_M^+, M(x - [d]_M^+) \right\rangle \leq 0 \qquad \forall x \in G, \forall d. \tag{13.1}$$

By substituting d by $x + d$ in (13.1) and simple algebra, we have

$$\left\langle Md, [x + d]_M^+ - x \right\rangle \geq \left\| x - [x + d]_M^+ \right\|_M^2 \qquad \forall x \in G, \forall d. \tag{13.2}$$

Applying (15.2) with

$$d = -\alpha M_w^{-1}(n) d_w(n),$$

$$M = M_w(n),$$

$$x = x_w(n),$$

$$G = X_w(n),$$

we get, by (2),

$$-\alpha \langle d_w(n), s_w(n) \rangle \geq \|s_w(n)\|_{M_w(n)}^2 \geq \delta \|s_w(n)\|^2.$$

Q.E.D.

The next lemma provides a bound on the first order terms in the Taylor series expansion of D.

Lemma 4.4 There exists a positive constant A_7, depending only on Δ and W, such that for all $w \in W$, for all $n = 0, 1, 2, \ldots$, for any $y \in X_w(n)$,

$$\langle d_w(n), x_w^*(n+1) - y \rangle \leq \frac{A_7}{\alpha} r_w(n) \| s_w(n) \|. \tag{14}$$

Proof. Following the same notation as that in Lemma 4.2, let

$$x(\alpha) = \left[x - \alpha d \right]_M^+.$$

By the definition of the projection, we have

$$\langle x - \alpha d - x(\alpha), M(y - x(\alpha)) \rangle \leq 0,$$

which implies

$$\langle x - x(\alpha), M(y - x(\alpha)) \rangle \leq \alpha \langle d, M(y - x(\alpha)) \rangle . \tag{15}$$

Applying (14) with

$$d = M_w^{-1}(n) d_w(n),$$

$$M = M_w(n),$$

$$x = x_w(n),$$

$$G = X_w(n),$$

we get

$$\begin{aligned} \langle d_w(n), x_w^*(n+1) - y \rangle &\leq \frac{1}{\alpha} \langle s_w(n), M(y - x_w^*(n+1)) \rangle \\ &\leq \frac{\Delta}{\alpha} \| s_w(n) \| \| y - x_w^*(n+1) \| . \end{aligned} \tag{16}$$

But $y, x_w^*(n+1) \in X_w(n)$ implies the existence of a scalar c depending on w, such that

$$\|y - x_w^*(n+1)\| \le c r_w(n). \tag{17}$$

Combining (16) and (17) gives (14). *Q.E.D.*

The following lemmas 4.3.a and 4.4.a (for the gradient projection algorithm with orthant constraints) correspond respectively to lemmas 4.3 and 4.4 (for the algorithm with simplex constraints).

Lemma 4.3.a There exists a positive constant A_8, depending only on W and δ, such that for all $w \in W$, for all $n = 0, 1, 2, \ldots$

$$< d_w(n), s_w(n) > \le -\frac{A_8}{\alpha} \|s_w(n)\|^2. \tag{18}$$

Proof. Note that $\forall p \in P_w, p \neq \overline{p}_w$,

$$0 \le -s_p(n) \le \frac{\alpha}{\hat{m}_{w,p}(n)} \left(d_p(n) - d_{\overline{p}_w}(n)\right). \tag{19}$$

Hence

$$-\sum_{p \in P_w \; p \neq \overline{p}_w} \left(d_p(n) - d_{\overline{p}_w}(n)\right) s_p(n) \ge \sum_{p \in P_w \; p \neq \overline{p}_w} |s_p(n)|^2 \frac{\hat{m}_{w,p}(n)}{\alpha},$$

which implies

$$\sum_{p \in P_w \; p \neq \overline{p}_w} \left(d_p(n) - d_{\overline{p}_w}(n)\right) s_p(n) \le -\frac{\delta}{\alpha} \sum_{p \in P_w \; p \neq \overline{p}_w} |s_p(n)|^2. \tag{20}$$

But

$$s_{\overline{p}_w}(n) = -\sum_{p \in P_w \; p \neq \overline{p}_w} s_p(n).$$

Thus by the Cauchy-Schwarz inequality,

$$\left|s_{\overline{p}_w}(n)\right|^2 \le |w| \|\hat{s}_w(n)\|^2, \tag{21}$$

where

$$\hat{s}_w(n) = \{s_p(n) \ : p \in P_w, p \neq \overline{p}_w\}.$$

Also

$$\sum_{p\in P_w \ p\neq \overline{p}_w} \left(d_p(n) - d_{\overline{p}_w}(n)\right) s_p(n) = \langle d_w(n), s_w(n)\rangle . \tag{22}$$

(20),(21) and (22) implies (18). *Q.E.D.*

Lemma 4.4.a There exists a positive constant A_9, depending only on Δ and W, such that for all $w \in W$, for all $n = 0, 1, 2, \ldots$, for any $y \in X_w(n)$,

$$\langle d_w(n), x^*_w(n+1) - y\rangle \leq \frac{A_9}{\alpha} r_w(n)\|s_w(n)\|. \tag{23}$$

Proof. Note that if $\hat{x}^*_p(n+1) = 0$, then

$$x_p(n) - \frac{\alpha}{\hat{m}_{w,p}(n)}\left(d_p(n) - d_{\overline{p}_w}(n)\right) - \hat{x}^*_p(n+1) \leq 0,$$

and if $\hat{x}^*_p(n+1) > 0$, then

$$x_p(n) - \frac{\alpha}{\hat{m}_{w,p}(n)}\left(d_p(n) - d_{\overline{p}_w}(n)\right) - \hat{x}^*_p(n+1) = 0.$$

This fact implies that

$$\left\langle \hat{x}_w(n) - \alpha\hat{M}_w^{-1}(n)\hat{d}_w(n) - \hat{x}^*_w(n+1), \hat{M}_w(n)\left(\hat{y} - \hat{x}^*_w(n+1)\right)\right\rangle \leq 0,$$

where

$$\hat{y} = \{y_p \ : p \in P_w, p \neq \overline{p}_w\}.$$

Thus, as in the proof of Lemma 4.4, we have

$$\left\langle -\hat{s}_w(n), \hat{M}_w(n)\left(\hat{y} - \hat{x}^*_w(n+1)\right)\right\rangle \leq \alpha\left\langle \hat{M}_w^{-1}(n)\hat{d}_w(n), \hat{M}_w(n)\left(\hat{y} - \hat{x}^*_w(n+1)\right)\right\rangle .$$

Hence

$$\begin{aligned}\left\langle \hat{d}_w(n), \hat{x}^*_w(n+1) - \hat{y}\right\rangle &= \langle d_w(n), x^*_w(n+1) - y\rangle \\ &\leq \frac{\Delta}{\alpha}\|\hat{s}_w(n)\|\|\hat{y} - \hat{x}_w(n+1)\|.\end{aligned}$$

(23) is true by noting the fact:

$$\|\hat{s}_w(n)\| \leq \|s_w(n)\|,$$

$$\|\hat{y} - \hat{x}_w(n+1)\| \leq cr_w(n),$$

for a positive constant c. *Q.E.D.*

4.5 The Convergence Result

The convergence result states that the average network cost decreases at a exponential rate which is bounded below by the slowest VC departure rate, to within a neighborhood of the long-term optimal cost. If the stepsize is small enough, the smaller the extent of violation of the ***many small users*** assumption the smaller the long-term deviation from optimality. The bound on long-term deviation from optimality, $\overline{\lim}_{n\to\infty} a(n, \varepsilon)$, is a function of the stepsize and the system parameter set $\mathbf{A}$, which includes the initial deviation from optimality, the update interval, the smallest VC departure rate, and etc.

There are two major differences between the convergence result (Theorem 4.5) and the counterpart (Theorem 3.1) of shortest path routing. First, the update interval needs not to be small in order to reduce the long-term deviation from optimality. One way to understand this property is to consider the update equations (2). With the stepsize reasonably small, the gradient projection algorithm will shift only moderate amounts of flow to the current shortest paths, during an update. Thus even if the update interval is large as compared to the average VC holding time, the large number of the new VCs arriving between the updates will not all be assigned to the shortest paths. Thus the problem of overloading the shortest paths is avoided.

We can also consider a routing update as a step in a stochastic programming algorithm. The algorithm moves the flows in a direction of decent based on the current linearization of the cost. If the step results in a large shift of flows, the linearization may not be good enough, the resulting flow would contribute to an increase in the cost. The shortest path algorithm with fast VC departure rate (as compared to the updating frequency) is committing the error of large step. On the other hand, the gradient projection algorithm *controls* the amount of flow shifting by adjusting its stepsize, essentially independently of the update interval (when the update interval is small, all the routing algorithms move a small step).

The second major difference between the two algorithms is that the gradient projection algorithms have an extra parameter, the stepsize, to fine-tune their performances. The result says that the long-term deviation from optimality is a function of the stepsize.

Theorem 4.5 There exist a positive constant, c_1, and a positive function $a(n,\varepsilon)$ (which depends only on the system parameter set $\mathbf{A}$ and α), such that $\forall n = 0, 1, 2, \ldots$

$$-c_1 e^{-\mu n T} \le E\left[D(n)\right] - D^* \le e^{-\mu n T}\left[D(0) - D^*\right] + a(n,\varepsilon), \tag{24.1}$$

$$\overline{\lim_{n\to\infty}}\, a(n,\varepsilon) < \infty. \tag{24.2}$$

Furthermore, there exists a scalar $\overline{\alpha} > 0$ such that $\forall \alpha \in (0, \overline{\alpha}], \forall \varepsilon \le 1$, the following is true:

(a)There exists a positive constant c_2 (which depends on the system parameter set $\mathbf{A}$ and α), such that

$$\overline{\lim_{n\to\infty}}\, E[D(n)] - D^* \le c_2 \varepsilon^{\frac{1}{4}}, \tag{25}$$

$$\lim_{\varepsilon\to 0} \overline{\lim_{n\to\infty}}\, E\|s(n)\|^2 = 0, \tag{26}$$

$$\lim_{\varepsilon\to 0} \overline{\lim_{n\to\infty}}\, a(n,\varepsilon) = 0, \tag{27}$$

$$\lim_{\varepsilon\to 0} \overline{\lim_{n\to\infty}}\, E[D(n)] = D^*. \tag{28}$$

(b) If, in addition, $\overline{D}_{ij}$ is strictly convex for all $(i,j) \in L$, then

$$\lim_{\varepsilon\to 0} \overline{\lim_{n\to\infty}}\, E\|F(n) - F^*\|^2 = 0.$$

The proof of Theorem 4.5 is fairly long and complicated, therefore we provide some intuition here. Although our model is in continuous time, we can concentrate on the discrete events of routing update.

Roughly speaking, our approach is to derive a linear first order dynamic system whose state is an upper bound on the average cost. If the linear system is stable with bounded input, we can compute the bound on the average cost easily by using the variation of constants formula. To derive this linear system, we linearize the cost at each update using the first order Taylor series expansion with remainder. So long as the average cost has a sufficient decrease in one routing update, after taking into account the second and the higher order terms, the system will be stable.

When the algorithm converges to optimality in the limit $\varepsilon \to 0$, the step $s(n)$ must go to zero in some sense. If $s(n)$ is zero, all the flows are on the shortest paths with respect to the FDLs, which is the optimality condition (see [Bertsekas 1982d]). By Lemmas 4.3 and 4.3.a, if the step is small enough there will be a sufficient decrease in the cost. If the algorithm converges, the cost cannot decrease forever, the step then must go to zero.

The proof is complicated by two factors. First, we have to ensure the effects of the second and the higher order terms in the Taylor series to be small. Second, the loads $r_w(n)$ are varying stochastically. Our approach is to estimate these uncertainties by bounding them and to show that these bounds only affect the main argument slightly.

Proof of Theorem 4.2. In this proof, the constants $A_1 - A_{40}$ are positive and depend only on the system parameter set $\mathbf{A}$ and perhaps α. We first show the inequality (24). The left hand side follows directly from the beginning of the proof of Theorem 1 in [GB].

To show the right hand side, take the Taylor series expansion:

$$D(n+1) \le D(n) + \sum_{p\in P} d_p(n)\left(x_p(n+1) - x_p(n)\right) + A_{10}\|x(n+1) - x(n)\|^2, \tag{29}$$

where A_{10} is a positive constant depending only on the system parameter set $\mathbf{A}$. Let

$$z_1(n) = \sum_{p\in P} d_p(n)\left(x_p(n+1) - x_p(n)\right), \tag{30.1}$$

$$z_2(n) = A_{10}\|x(n+1) - x(n)\|^2. \tag{30.2}$$

First we bound $z_1(n)$ by using (8)

$$\begin{aligned}
&E\left[z_1(n)|x(n)\right]\\
&= \sum_{w\in W}\left(1 - e^{-\mu_w T}\right)\langle d_w(n), \tilde{x}_w^*(n+1) - x_w(n)\rangle\\
&= \sum_{w\in W}\left(1 - e^{-\mu_w T}\right)\left[\langle d_w(n), x_w^*(n+1) - x_w(n)\rangle + \langle d_w(n), \tilde{x}_w^*(n+1) - x_w^*(n+1)\rangle\right].
\end{aligned} \tag{31}$$

By Lemma 4.3

$$\langle d_w(n), x_w^*(n+1) - x_w(n)\rangle \le 0.$$

Thus (31) can be strengthened to yield

$$E\left[z_1(n)|x(n)\right] \le z_3(n) + z_4(n), \tag{32.1}$$

where

$$z_3(n) = \left(1 - e^{-\mu T}\right)\sum_{p\in P} d_p(n)\left(x_p^*(n+1) - x_p(n)\right), \tag{32.2}$$

$$z_4(n) = \sum_{w\in W}\left(1 - e^{-\mu_w T}\right)\langle d_w(n), \tilde{x}_w^*(n+1) - x_w^*(n+1)\rangle. \tag{32.3}$$

Let $x^* = \{x_w^* : w \in W\}$ be any optimal deterministic routing, i.e.,

$$D(x^*) = D^*.$$

Let

$$\overline{x}(n) = \{\overline{x}_w(n) \, : \overline{x}_w(n) = \frac{x_w^* r_w(n)}{\overline{r}_w}, \forall w \in W\}.$$

Hence $\overline{x}_w \in X_w(n)$, the simplex for the OD pair w at time nT. By adding and subtracting terms, transform (32) into

$$E\left[z_1(n)|x(n)\right] \leq z_5(n) + z_6(n) + z_7(n) + z_4(n), \tag{33.1}$$

where

$$z_5(n) = \left(1 - e^{-\mu T}\right) \sum_{p \in P} d_p(n) \left[x_p^*(n+1) - \overline{x}_p(n)\right], \tag{33.1}$$

$$z_6(n) = \left(1 - e^{-\mu T}\right) \sum_{p \in P} d_p(n) \left[x_p^* - x_p(n)\right], \tag{33.2}$$

$$z_7(n) = \left(1 - e^{-\mu T}\right) \sum_{w \in W} < d_w(n), x_w^* > \left(\frac{r_w(n) - \overline{r}_w}{\overline{r}_w}\right). \tag{33.3}$$

We shall now bound $z_4(n) - z_7(n)$. First, observe that

$$\begin{aligned} \|d_w\left(x(n)\right)\| \leq & \|d_w\left(x(n)\right) - d_w(0)\| + \|d_w(0)\| \\ \leq & A_{11} \sum_{w \in W} \|x_w(n)\| + A_{12} \\ \leq & A_{13} \sum_{w \in W} r_w(n) + A_{12}. \end{aligned} \tag{34}$$

By (7)

$$E\left[r_w(n)\right] \leq A_{14}, \tag{35}$$

$$E\left[r_w^2(n)\right] = \text{Var}\left[r_w(n)\right] + \left(E\left[r_w(n)\right]\right)^2 \leq A_{15}, \tag{36}$$

$$\begin{aligned} E\left[|\overline{r}_w - r_w(n)|^2\right] =& \text{Var}\left[r_w(n)\right] + \left(E\left[r_w(n)\right] - \overline{r}_w\right)^2 \\ \leq & A_3\varepsilon + A_4 e^{-\mu n T}. \end{aligned} \tag{37}$$

Now by Lemma 4.4, the Hölder's inequality and (36)

$$\begin{aligned} E\left[z_5(n)\right] \leq & \frac{A_7}{\alpha} E\left[\sum_{w \in W} r_w(n) \|s_w(n)\|\right] \\ \leq & \frac{A_7}{\alpha} \sum_{w \in W} \left(E\left[r_w^2(n)\right]\right)^{\frac{1}{2}} \left(E\left[\|s_w(n)\|^2\right]\right)^{\frac{1}{2}} \\ \leq & \frac{A_{16}}{\alpha} \sum_{w \in W} \left(E\left[\|s_w(n)\|^2\right]\right)^{\frac{1}{2}}. \end{aligned} \tag{38}$$

By the convexity of D

$$z_6(n) \le \left(1 - e^{-\mu T}\right)\left(D^* - D(n)\right). \tag{39}$$

Using the Hölder's inequality, (35) and (37)

$$\begin{aligned} E\left[z_7(n)\right] &\le A_{17} E\left[\sum_{w\in W} |\bar{r}_w - r_w(n)| \|d_w(n)\| \|x_w^*\|\right] \\ &\le A_{17} \sum_{w\in W} \left(E|\bar{r}_w - r_w(n)|^2\right)^{\frac{1}{2}} \left(E\left[A_{18} \sum_{w\in W} r_w(n) + A_{19}\right]^2\right)^{\frac{1}{2}} \\ &\le A_{20}\left[A_3\varepsilon + A_4 e^{-\mu n T}\right]^{\frac{1}{2}}. \end{aligned} \tag{40}$$

Similarly by (34), (10.2) in Lemma 4.2, and (36)

$$E\left[z_4(n)\right] \le A_{21}\left[A_3\varepsilon + A_4 e^{-\mu n T}\right]^{\frac{1}{2}}. \tag{41}$$

By (38)-(41), we now have

$$\begin{aligned} E\left[z_1(n)\right] \le& \left(1 - e^{-\mu T}\right)\left[D^* - E\left(D(n)\right)\right] \\ &+ A_{22}\left[A_3\varepsilon + A_4 e^{-\mu n T}\right]^{\frac{1}{2}} + \frac{A_{16}}{\alpha} \sum_{w\in W} \left(E\left[\|s_w(n)\|^2\right]\right)^{\frac{1}{2}}. \end{aligned} \tag{42}$$

By (10.3) in Lemma 4.2

$$E\left[z_2(n)\right] \le A_{23}\varepsilon + A_{24} E\left[\|s(n)\|^2\right] + A_{25} e^{-\mu n T}. \tag{43}$$

Combining (42), (43), and (29)

$$E\left[D(n+1)\right] \le E\left[D(n)\right] + \left(1 - e^{-\mu T}\right)\left[D^* - E\left(D(n)\right)\right] + b(n,\varepsilon), \tag{44.1}$$

where

$$\begin{aligned} b(n,\varepsilon) =& A_{23}\varepsilon + A_{24} E\left[\|s(n)\|^2\right] + A_{25} e^{-\mu n T} + A_{22}\left[A_3\varepsilon + A_4 e^{-\mu n T}\right]^{\frac{1}{2}} \\ &+ \frac{A_{16}}{\alpha} \sum_{w\in W} \left(E\left[\|s_w(n)\|^2\right]\right)^{\frac{1}{2}}. \end{aligned} \tag{44.2}$$

By repeated application of (44), we get

$$E\left[D(n)\right] - D^* \leq e^{-\mu n T}\left[E\left[D(0)\right] - D^*\right] + a(n,\varepsilon), \tag{45.1}$$

where

$$a(n,\varepsilon) = \sum_{i=1}^{n-1} b(i,\varepsilon)e^{-\mu(n-i-1)T}. \tag{45.2}$$

Since (44.1) is a stable linear system, and $b(n,\varepsilon)$ is bounded, the limit

$$\overline{\lim_{n\to\infty}} a(n,\varepsilon)$$

exists. Thus completes the proof of (24).

To show (a) By Lemma 4.3

$$z_3(n) \leq -\frac{A_{28}}{\alpha}\|s(n)\|^2. \qquad (46).$$

By Lemma 4.2 (10.1)

$$E\left[z_2(n)|x(n)\right] \leq A_{28}\varepsilon + A_2\left[\|s(n)\|^2 + \|\tilde{x}^*(n+1) - x^*(n+1)\|^2\right]. \tag{47}$$

Thus (41), (48), (49), (29) imply

$$E\left[D(n+1)|x(n)\right] \leq D(n) - \left(\frac{A_{28}}{\alpha} - A_2\right)\|s(n)\|^2 + z_8(n), \tag{48.1}$$

with

$$\begin{aligned} z_8(n) =& A_{29}\sum_{w\in W}|\langle d_w(n), \tilde{x}^*_w(n+1) - x^*_w(n+1)\rangle| \\ &+ A_{28}\varepsilon + A_{30}\|\tilde{x}^*(n+1) - x^*(n+1)\|^2 > 0. \end{aligned} \tag{48.2}$$

Choose α small enough so that

$$A(\alpha) = \left(\frac{A_{28}}{\alpha} - A_2\right) > 0.$$

Use convexity to get

$$D(n) + \sum_{w \in W} \langle d_w(n), x_w^* - x_w(n) \rangle \le D^*.$$

Hence

$$\begin{aligned} D(n) - D^* &\le \sum_{w \in W} \langle d_w(n), x_w(n) - x_w^*(n) \rangle \\ &= \sum_{w \in W} \{ \langle d_w(n), x_w(n) - x_w^*(n+1) \rangle + \langle d_w(n), x_w^*(n+1) - \bar{x}_w(n) \rangle \\ &+ \langle d_w(n), \bar{x}_w(n) - x_w^* \rangle \}, \end{aligned}$$

where

$$x_w(n) = \frac{x_w^* \bar{r}_w(n)}{\bar{r}_w}$$

is a vector in

$$X_w(n) = \{x_w : \sum_{p \in P_w} x_p = r_w(n), x_p \ge 0\}.$$

Note that

$$\langle d_w(n), x_w(n) - x_w^*(n+1) \rangle \le \|d_w(n)\| \|s_w(n)\|.$$

By Lemma 4.4

$$\langle d_w(n), x_w^*(n+1) - \bar{x}_w(n) \rangle \le \frac{A_7}{\alpha} r_w(n) \|s_w(n)\|,$$

$$\langle d_w(n), \bar{x}_w(n) - x_w^* \rangle \le \frac{\|d_w(n)\| \|x_w^*\| |r_w(n) - \bar{r}_w|}{\bar{r}_w}.$$

These inequalities lead to

$$E[D(n)] - D^* \le A_{31} \left(E\|s(n)\|^2 \right)^{\frac{1}{2}} + A_{32} \left[A_5 \varepsilon + A_6 e^{-\mu n T} \right]^{\frac{1}{2}}.$$

Thus $\exists n_1 > 0$ such that $\forall n \ge n_1$

$$E[D(n)] - D^* \le A_{33} \left(E\|s(n)\|^2 \right)^{\frac{1}{2}} + A_{31} \varepsilon^{\frac{1}{2}}. \tag{49}$$

Rewrite (49) into, with $A_{34} > 0$, and $A_{35} > 0$, which depend on α,

$$A_{34} \{ E[D(n)] - D^* \} - A_{35} \varepsilon^{\frac{1}{2}} \le \left[A(\alpha) E\|s(n)\|^2 \right]^{\frac{1}{2}}. \tag{50}$$

We can bound (48) by, using Lemma 4.2,

$$E[D(n+1)] \leq [D(n)] - A(\alpha)E\|s(n)\|^2 + A_{36}\varepsilon + A_{37}e^{-\mu nT}$$
$$+ A_{38}\left(\left[A_3\varepsilon + A_4 e^{-\mu nT}\right]^{\frac{1}{2}}\right).$$

Now, by (48), there exists $n_2 \geq n_1$ such that $\forall n \geq n_2$, $\forall \varepsilon \leq 1$

$$E[D(n+1)] \leq E[D(n)] - A(\alpha)E\|s(n)\|^2 + A_{39}\varepsilon^{\frac{1}{2}}. \tag{51}$$

Let $A_{40} = (A_{39})^{\frac{1}{2}}$. Suppose that, for all $n \geq n_2$,

$$E[D(n)] - D^* \geq (A_{34})^{-1}\{(A_{35}+1)\varepsilon^{\frac{1}{2}} + A_{40}\varepsilon^{\frac{1}{4}}\},$$

then

$$A_{34}\{E[D(n)] - D^*\} - A_{35}\varepsilon^{\frac{1}{2}} \geq \varepsilon^{\frac{1}{2}} + A_{40}\varepsilon^{\frac{1}{4}}.$$

By (50)

$$A(\alpha)E\|s(n)\|^2 \geq (\varepsilon^{\frac{1}{2}} + A_{40}\varepsilon^{\frac{1}{4}})^2 \geq \varepsilon + A_{39}\varepsilon^{\frac{1}{2}},$$

which implies

$$E[D(n+1)] - E[D(n)] \leq \varepsilon.$$

The above facts now imply that

$$\overline{\lim_{n\to\infty}} E[D(n)] - D^* \leq (A_{34})^{-1}\left((A_{35}+1)\varepsilon^{\frac{1}{2}} + A_{40}\varepsilon^{\frac{1}{4}}\right) + A_{39}\varepsilon^{\frac{1}{2}},$$

which is essentially (25).

(28) follows trivially from (25).

By (51),

$$E\|s(n)\|^2 \leq (A(\alpha))^{-1}\left\{E[D(n) - E[D(n+1)] + A_{39}\varepsilon^{\frac{1}{2}}\right\}.$$

Thus

$$\lim_{\varepsilon\to 0}\overline{\lim_{n\to\infty}} E\|s(n)\|^2 = 0,$$

which implies (27).

(b) is proved by the end of the proof of Theorem 3.1. *Q.E.D.*

CHAPTER FIVE
ASYNCHRONOUS GRADIENT PROJECTION ROUTING WITH RANDOMIZED PATH ASSIGNMENT

5.1 Introduction

The convergence result of distributed asynchronous gradient projection routing with randomized path assignment is presented in this chapter.

We first discuss the fundamental issues of asynchronous routing and the approach used to show the convergence result in section 5.2. We also present an example showing the non-convergence of the routing algorithm, under the condition that the time between consecutive receptions of flow measurements is unbounded.

Some preliminary lemmas which are needed to prove the convergence result are presented in section 5.3. Lemma 5.2 gives the descent property of the gradient projection algorithm. Other lemmas are the estimates about the uncertainities in the routing environment.

The convergence result, its proof, and interpretations are presented in section 5.4. The gradient projection algorithms we consider in this chapter are similar to the algorithms of chapter 4 with either the simplex constraints (equation (1) of chapter 4) or the orthant constraints (equation (3) of chapter 4). The only difference is that the FDLs $d_w(n)$ are now replaced by the estimated FDLs $\eta_w(n)$. We shall only present the proof for the algorithm with the simplex constraints– the proof for the algorithm with the orthant constraints is only a slight modification of the proof presented in this chapter.

5.2 Fundamental Issues Of Asynchronous Routing

There are several reasons why practical routing should be asynchronous. First, the routing should adapt to sudden large flow fluctuations. At a sudden surge of local congestion, the routing controller should have the freedom to perform an update in an effort to reduce this local problem, without waiting till its next designated update time. Second, the forwarding of control packets (containing flow measurements) has unpredictable delays. If the delays are predictable, a global controller can order the updating times for each local controller, in an effort to solve the global optimization problem with a strict coordination among the local controllers.

The asynchronous case is more complicated than the synchronous case in many aspects. First, in the asynchronous model we do not assume perfect measurement of link flows. As a result, the FDLs for all links, which are needed for computing the routing variables at an update, is usually inaccurate. Furthermore, the link flow measurements will be forwarded to all nodes that need this information, with a bounded delay. Also the routing controllers for each OD pairs perform updates not in synchronism with each other. All these particularities add to the complexity of the analysis.

In practice, each routing controller at the nodes of the network has its own local memory storing the information needed to perform a routing update. These controllers together are to solve a global optimization problem to attain an optimal routing. The first problem that arises is how to guarantee that their information is mutually consistent so that they are together solving the same global problem. This problem is in fact a kind of *agreement problem* in fault-tolerant computing.

For quasi-static routing, this agreement problem causes little concern for us. We assume that the network is reliable and all the link flow measurements are broadcast to all nodes in the network with a bounded delay. Such a broadcast mechanism ensures that each controller has approximately the same information. If all the controllers stop updating and the load is kept constant with the broadcast mechanism running, the flow estimates at each node will agree with each other in a bounded time. Thus, the agreement problem is in a sense trivially solved.

The proof technique for showing the convergence result is that of [TB]. Two keys of this approach are the small stepsize and the assumption that the time between consecutive receptions of flow measurements is bounded.

The Need For A Small Stepsize

In the synchronous case, the main reason that the stepsize must be small is that a small stepsize ensures the linearization of the cost at each routing update to be good. In the asynchronous case, there is one additional reason for a small stepsize: the flow estimates at each routing updates are inaccurate and possibly quite out-of-date. If the algorithm moves the flows slowly then the flow estimates at each node will become more accurate and up-to-date. With a more accurate flow estimate, the algorithm has a better chance of decreasing the cost at each update.

The Need For A Bounded Time Between Measurement Receptions

The problem with an unbounded time between consecutive measurement receptions is that the flow estimates available at local nodes may be far out-of-date. We provide here an example, which is simpler than that in [TB], to show the non-convergence of the algorithm without this boundedness assumption. Consider the following simple network. There are two OD pairs: $w_1 = (1,4), w_2 = (2,4)$, each with two paths, one via link a_3, the other via link a_4. Assume that both links a_3 and a_4 have a capacity of 2.01, and both links a_1 and a_2 have a capacity of 10.0, each OD pair has an input traffic of 1.0. Suppose the link cost $\overline{D}_a$ for a link a is given by the average queue size for an $M/M/1$ queue:

$$\overline{D}_a(F_a) = \frac{F_a}{C_a - F_a},$$

where F_a and C_a are the link flow and the capacity of the link respectively.

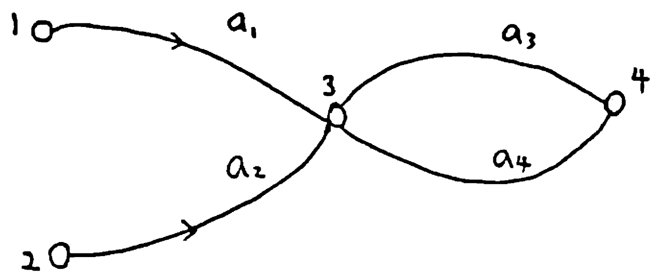


Figure 5.1 The Simple Counter-Example Network

It is obvious that an optimal routing must have both links a_3 and a_4, each carries a flow of 1.0. Suppose that the initial loading is that both OD pairs send all their traffics through link a_3. Let node 1 and 2 both carry out a large number of gradient projection updates without receiving any new flow measurement updates. Assuming fast VC departure and arrival rates, at the time of next measurement reception, all the traffic will be going through link a_4. If the nodes receive the same measurements, and the next measurement updates and receptions are again very late, all the flows will be sent through link a_3. In this manner, the routing will oscillate just like a shortest path routing with long update intervals, and convergence to optimality is impossible.

5.3 Preliminary Lemmas

As the approach of proving convergence of the asynchronous algorithm is similar to that of the synchronous version, we need the corresponding counterparts of Lemmas 4.3, and 4.4. The first lemma provides the descent property of a step in the gradient projection algorithm, while the second lemma gives a bound on the first order terms in the Taylor series expansion.

To deal with the particularities in the asynchronous case we need to bound the extra uncertainities in the network. These uncertainities include inaccurate flow measurements and delays due to broadcasting. The lemmas 5.3 and 5.4 serve this purpose.

Lemma 5.1 (c.f. Lemma 4.3) For all $w \in W$ and for all $n \in T_w$,

$$\langle \eta_w(n), s_w(n) \rangle \leq -\frac{\delta}{\alpha}\|s_w(n)\|^2. \tag{1}$$

Lemma 5.2 (c.f. Lemma 4.4) There exists a positive constant A_1, depending only on Δ and W such that for all $w \in W$, for all $n \in T_w$, for any $y \in X_w(n)$,

$$\langle \eta_w(n), x_w^*(n+1) - y \rangle \leq \frac{A_1}{\alpha} r_w(n)\|s_w(n)\|. \tag{2}$$

The proof of Lemmas 5.1 and 5.2 are omitted because they can be derived by simply modifying the proofs of Lemmas 4.3 and 4.4.

Lemma 5.3 There exist positive constants $A_2 - A_6$ depending only on the system parameter set $\mathbf{A}$, such that $\forall n = 0, 1, 2, \ldots$, $\forall w \in W$

$$E[\|x_w(n+1) - x_w(n)\|^2] \leq A_2\varepsilon + A_3 E\|s_w(\overline{n}_w(n))\|^2 + A_4 e^{-2\mu n T}, \tag{3}$$

$$E[\|x_w^*(n+1) - \tilde{x}_w^*(n+1)\|^2] \leq A_5\varepsilon + A_6 e^{-2\mu n T}, \tag{4.1}$$

$$E[|r_w(n) - \overline{r}_w|^2] \leq A_5\varepsilon + A_6 e^{-2\mu n T}. \tag{4.2}$$

Proof: Fix any w we have

$$\|x_w(n+1) - x_w(n)\|^2 \leq 2\|x_w(n+1) - \tilde{x}_w^*(n+1)\|^2 + 2\|x_w(n) - \tilde{x}_w^*(n+1)\|^2.$$

For all $p \in P_w$,

$$\begin{aligned}
&E\left[|x_p(n) - \tilde{x}_p^*(n+1)|^2 \;\middle|\; x_w(\overline{n}_w(n))\right] \\
=&\mathrm{Var}[x_p(n)|x_w(\overline{n}_w(n))] + [E\{x_p(n)|x_w(\overline{n}_w(n))\} - \tilde{x}_p^*(n+1)]^2 \\
=&\varepsilon\overline{\gamma}_w(1 - e^{-\mu_w T(n-\overline{n}_w(n))})[\tilde{x}_p^*(n+1) + e^{-\mu_w T(n-\overline{n}_w(n))}x_p(\overline{n}_w(n))] \\
&+ (e^{-\mu_w T(n-\overline{n}_w(n))})^2(x_p(\overline{n}_w(n)) - \tilde{x}_p^*(n+1))^2 \\
\leq&\varepsilon\overline{\gamma}_w(\tilde{x}_p^*(n+1) + x_p(\overline{n}_w(n))) + 2|s_p(\overline{n}_w(n))|^2 + 2\left|\tilde{x}_p^*(n+1) - x_p^*(n+1)\right|^2.
\end{aligned}$$

Hence

$$E[\|x_w(n) - \tilde{x}_w^*(n+1)\|^2] \le A_7\varepsilon + 2E\|s_w(\overline{n}_w(n))\|^2 + A_8 e^{-2\mu n T}.$$

Similarly

$$E[\|x_w(n) - \tilde{x}_w^*(n+1)\|^2] \le A_7\varepsilon + 2E\|s_w(\overline{n}_w(n))\|^2 + A_8 e^{-2\mu n T}.$$

Now (3) follows. (4) follows immediately from Lemma 4.1. *Q.E.D.*

Lemma 5.4 There exist positive constants $A_9 - A_{11}$ such that for all OD pairs $w \in W$, and for all $n = 0, 1, 2, \ldots$,

$$\left\|\frac{\partial D}{\partial x_w}(n) - \eta_w(n)\right\| \le A_9 \sum_{m=1}^{C} \|x(n+m-1) - x(n+m)\|, \tag{5}$$

$$\|\eta_w(n)\| \le A_{10} \sum_{m=1}^{C} \sum_{w\in W} r_w(n-m) + A_{11}. \tag{6}$$

Proof:

$$\begin{aligned}
\left\|\frac{\partial D}{\partial x_w}(n) - \eta_w(n)\right\| &= \left\|\frac{\partial D}{\partial x_w}(n) - \eta_w(\overline{n}_w(n))\right\| \\
&\le A_{12} \max_{i,j,k} |\hat{F}_{ij,k}(\overline{n}_w(n)) - F_{ij}(n)| \\
&\le A_{13} \max_{i,j} \max_{n-C\le m\le n} \|F_{ij}(m) - F_{ij}(n)| \\
&\le A_9 \max_{n-C\le m\le n} \|x(m) - x(n)\| \\
&\le A_9 \sum_{m=1}^{C} \|x(n-m+1) - x(n-m)\|,
\end{aligned}$$

which is (5). To show (6),

$$\begin{aligned}
\|\eta_w(n)\| &\le \left\|\eta_w(\overline{n}_w(0)) - \frac{\partial D}{\partial x_w}(0)\right\| + \left\|\frac{\partial D}{\partial x_w}(0)\right\| \\
&\le A_{12} \max_{i,j,k} |\hat{F}_{ij,k}(\overline{n}_w(n)) - 0| + A_{11} \\
&\le A_{13} \max_{i,j} \max_{n-C\le m\le n} |F_{ij}(m) - 0| + A_{11} \\
&\le A_9 \max_{n-C\le m\le n} \|x(m)\| + A_{11} \\
&\le A_{10} \sum_{m=1}^{C} \sum_{w\in W} r_w(n-m) + A_{11}.
\end{aligned}$$

Q.E.D.

5.4 The Convergence Result

The convergence result is similar to that of the synchronous case. The major difference is the convergence rate estimate.

The convergence rate estimate clearly reflects the effect of asynchronism. We see that the convergence rate estimate $\beta(\overline{n})$ satisfies

$$e^{-\mu T} \leq \beta(\overline{n}) < 1, \qquad \forall n = 0, 1, 2, \ldots$$

Also $\beta(\overline{n})$ increases to 1, or the convergence rate becomes slower, as $\overline{n}$, the bound on time between consecutive updates, goes to infinity. This is intuitively obvious. As the routing updates become slower, it takes longer to advance to optimality.

Note that the long-term deviation from optimality is now a function of C, a bound on measurement periods, delays in control packets and time between consecutive routing updates. The proof reveals that the larger the C, the larger the long-term deviation.

Theorem 5.5 There exist a positive constant, c_1, and a positive function $a(n, \varepsilon)$ (which depends only on the system parameter set $\mathbf{A}$ and α), such that $\forall n = 0, 1, 2, \ldots$

$$-c_1 e^{-\mu n T} \leq E\left[D(n)\right] - D^* \leq \beta(\overline{n})\left[D(0) - D^*\right] + a(n, \varepsilon), \tag{7.1}$$

$$\beta(\overline{n}) = 1 - e^{-MT\overline{n}}\left(1 - e^{-\mu T}\right), \tag{7.2}$$

$$\overline{\lim_{n\to\infty}}\, a(n, \varepsilon) < \infty. \tag{7.3}$$

Furthermore, there exists a scalar $\overline{\alpha} > 0$ such that $\forall \alpha \in (0, \overline{\alpha}], \forall \varepsilon \leq 1$, the following is true:

(a)There exists a positive constant c_2 (which depends on the system parameter set $\mathbf{A}$ and α) such that

$$\overline{\lim_{n\to\infty}} E[D(n)] - D^* \leq c_2 \varepsilon^{\frac{1}{4}}, \tag{8}$$

$$\lim_{\varepsilon\to 0}\overline{\lim_{n\to\infty}} E\|s_w(\overline{n}_w(n))\|^2 = 0, \forall w \in W, \tag{9}$$

$$\lim_{\varepsilon\to 0}\overline{\lim_{n\to\infty}} a(n,\varepsilon) = 0, \tag{10}$$

$$\lim_{\varepsilon\to 0}\overline{\lim_{n\to\infty}} E[D(n)] = D^*. \tag{11}$$

(b) If, in addition, $\overline{D}_{ij}$ is strictly convex for all $(i,j) \in L$, then

$$\lim_{\varepsilon\to 0}\overline{\lim_{n\to\infty}} E\|F(n) - F^*\|^2 = 0.$$

Proof:

First we show inequality (7). The left-hand-side is shown by the beginning of the proof of Theorem 1 of [GB]. To show the right-hand side, take the Taylor series expansion:

$$D(n+1) \le D(n) + \sum_{w\in W} < d_w(n), x_w(n+1) - x_w(n) > +A_{14}\|x(n+1) - x(n)\|^2.$$

Rewrite the above inequality into

$$D(n+1) \le D(n) + z_1(n) + z_3(n),$$

where

$$\begin{aligned} z_1(n) &= \sum_{w\in W} < \eta_w(n), x_w(n+1) - x_w(n) >, \\ z_2(n) &= \sum_{w\in W} < d_w(n) - \eta_w(n), x_w(n+1) - x_w(n) >, \\ z_3(n) &= A_{14}\|x(n+1) - x(n)\|^2. \end{aligned}$$

By Lemma 4.1,

$$E[x_w(n+1)|x_w(\overline{n}_w(n))] = \tilde{x}^*_w(n+1) + e^{-\mu_w T(n-\overline{n}_w(n)+1)}(x_w(\overline{n}_w(n)) - \tilde{x}^*_w(n+1)),$$

$$E[x_w(n)|x_w(\overline{n}_w(n))] = \tilde{x}^*_w(n+1) + e^{-\mu_w T(n-\overline{n}_w(n))}(x_w(\overline{n}_w(n)) - \tilde{x}^*_w(n+1)).$$

Hence

$$\begin{aligned} E[z_1(n)] =&E[\sum_{w\in W}(< \eta_w(n), \tilde{x}^*_w(n+1) - x_w(\overline{n}_w(n)) > \\ &+ < \eta_w(n), \tilde{x}^*_w(n+1) - x_w(\overline{n}_w(n)) >)e^{-\mu_w T(n-\overline{n}_w(n))}(1 - e^{-\mu_w T})]. \end{aligned}$$

With the above expression for $E[z_1(n)]$, we can write

$$E[z_1(n)] = E[z_4(n)] + E[z_5(n)],$$

where

$$z_4(n) = \sum_{w \in W} < \eta_w(n), x_w^*(n+1) - x_w(\overline{n}_w(n)) > e^{-\mu_w T(n-\overline{n}_w(n))}(1 - e^{-\mu_w T}),$$

$$z_5(n) = \sum_{w \in W} < \eta_w(n), x_w^*(n+1) - x_w(n+1) > e^{-\mu_w T(n-\overline{n}_w(n))}(1 - e^{-\mu_w T}).$$

Now by Lemma 5.1 (3), $\forall w \in W$,

$$< \eta_w(n), x_w^*(n+1) - x_w(\overline{n}_w(n)) > \leq 0.$$

Hence $z_4(n)$ can be bounded by

$$z_4(n) \leq \left(1 - e^{-\mu T}\right) e^{-MT\overline{n}} \sum_{w \in W} < \eta_w(n), x_w^*(n+1) - x_w(\overline{n}_w(n)) >,$$

where we recall

$$\mu = \min_w \{\mu_w\},$$

$$M = \max_w \{\mu_w\}.$$

$z_4(n)$ is still not in the form that can be easily bounded. Thus we add and subtract terms to get

$$z_4(n) \leq \left(1 - e^{-\mu T}\right) e^{-MT\overline{n}} [z_6(n) + z_7(n) + z_8(n) + z_9(n)],$$

where

$$z_6(n) = \sum_{w \in W} < \eta_w(n), x_w^*(n+1) - \overline{x}_w(n) >,$$

$$z_7(n) = \sum_{w \in W} < \eta_w(n), x_w^* - x_w(n) >,$$

$$z_8(n) = \sum_{w \in W} < \eta_w(n), \overline{x}_w(n) - x_w^* >,$$

$$z_9(n) = \sum_{w \in W} < \eta_w(n), x_w(n) - x_w(\overline{n}_w(n)) >,$$

x^* is any deterministic optimal routing, i.e.,

$$D(x^*) = D^*,$$

and

$$\overline{x}_w(n) = \frac{x^*_w r_w(\overline{n}_w(n)))}{\overline{r}_w}.$$

We further expand $z_7(n)$:

$$z_7(n) = z_{10} + z_{11}(n),$$

where

$$z_{10} = \sum_{w \in W} < d_w(n), x^*_w - x_w(n) >,$$

$$z_{11} = \sum_{w \in W} < \eta_w(n) - d_w(n), x^*_w - x_w(n) > .$$

Now we shall bound $E[z_i(n)]$, for $i = 2, 3, 5, 6, 8, 9, 10, 11$. First, we bound $E[z_i(n)]$ for $i = 6, 8, 5, 10$. By Lemma 5.2 (2)

$$z_6(n) \leq \frac{A_1}{\alpha} \sum r_w(n) \|s_w(\overline{n}_w(n))\|.$$

By the Hölder's inequality

$$E[z_6(n)] \leq \frac{A_{15}}{\alpha} \sum_{w \in W} (E\|s_w(\overline{n}_w(n))\|^2)^{\frac{1}{2}}. \tag{12}$$

Note that

$$\begin{aligned} z_8(n) &= \sum_{w \in W} < \eta_w(n), x^*_w > \frac{r_w(\overline{n}_w(n)) - \overline{r}_w}{\overline{r}_w} \\ &\leq \sum_{w \in W} \|\eta_w(n)\| \|x^*_w\| \frac{|r_w(\overline{n}_w(n)) - \overline{r}_w|}{\overline{r}_w}. \end{aligned}$$

By (2) and the Hölder's inequality, we get

$$E[z_8(n)] \leq \sum_{w \in W} A_{16}[A_5\varepsilon + A_6 e^{-2\mu \overline{n}_w(n) T}]^{\frac{1}{2}}. \tag{13}$$

Similarly

$$z_5(n) \leq (1 - e^{-MT}) \sum_{w \in W} \|\eta_w(n)\| \|\tilde{x}^*_w(n+1)\| \frac{|r_w(\overline{n}_w(n)) - \overline{r}_w|}{\overline{r}_w},$$

which implies

$$E[z_5(n)] \le \sum_{w \in W} A_{17}[A_5\varepsilon + A_6 e^{-2\mu\overline{n}_w(n)T}]^{\frac{1}{2}}. \tag{14}$$

By the convexity of D,

$$E[z_{10}(n)] \le D^* - E[D(n)]. \tag{15}$$

Now we bound $E[z_2(n)]$ and $E[z_{11}(n)]$

$$\begin{aligned} z_2(n) &= \sum_{w \in W} < d_w(n) - \eta_w(n), x_w(n+1) - x_w(n) > \\ &\le \sum_{w \in W} \|d_w(n) - tw(n)\| \|x_w(n+1) - x_w(n)\| \\ &\le A_{18} \sum_{m=1}^{C} \|x(n-m+1) - x(n-m)\| \|x(n+1) - x(n)\| \\ &\le \frac{1}{2} A_{18} \sum_{m=1}^{C} \|x(n-m+1) - x(n-m)\|^2 + \|x(n+1) - x(n)\|^2 \\ &\le A_{19} \sum_{m=0}^{C} \|x(n-m+1) - x(n-m)\|^2, \end{aligned}$$

where

$$A_{19} = \frac{A_{18}C}{2},$$

and the second inequality comes from Lemma 5.4.

Now by Lemma 5.3,

$$E[z_2(n)] \le A_{20}\varepsilon + A_{21}e^{-2\mu nT} + A_{22} \sum_{m=0}^{C} \sum_{w \in W} E\|s_w(\overline{n}_w(n-m))\|^2. \tag{16}$$

By the Hölder's inequality,

$$E[z_{11}(n)] \le \sum_{w \in W} \left(E\|\eta_w(n) - d_w(n)\|^2\right)^{\frac{1}{2}} \left(E\|x_w^* - x_w(n)\|^2\right)^{\frac{1}{2}}.$$

Note that $\exists A_{23} > 0$, such that $w \in W$, $p \in P_w$,

$$\begin{aligned} E[|x_p^* - x_p(n)|^2] &\le 2E[(x_p^*)^2 + r_w^2(n)] \\ &\le A_{23}. \end{aligned}$$

Hence, using Lemma 5.4,

$$
\begin{aligned}
E[z_{11}(n)] &\leq A_{24} \sum_{w \in W} \left(E\|\eta_w(n) - d_w(n)\|^2\right)^{\frac{1}{2}} \\
&\leq A_{25} \left(E\left[\left(\sum_{m=1}^{C} \|x(n-m+1) - x(n-m)\|\right)^2\right]\right)^{\frac{1}{2}} \\
&\leq A_{25} C \left(E\left[\sum_{m=1}^{C} \|x(n-m+1) - x(n-m)\|^2\right]\right)^{\frac{1}{2}} \\
&\leq A_{26} \left(A_4 \sum_{m=1}^{C} \sum_{w \in W} E\|s_w(\overline{n}_w(n-m))\|^2 + A_2 \varepsilon + A_4 e^{-2\mu n T}\right)^{\frac{1}{2}}, \qquad (17)
\end{aligned}
$$

where the third inequality comes from the Cauchy-Schwarz inequality.

Lastly, we bound $E[z_3(n)]$ and $E[z_9(n)]$. By Lemma 5.3,

$$
E[z_3(n)] \leq A_{14} \sum_{w \in W} \left(A_3 E\|s_w(\overline{n}_w(n))\|^2 + A_2 \varepsilon + A_4 e^{-2\mu n T}\right)^{\frac{1}{2}}. \qquad (18)
$$

Also

$$
\begin{aligned}
&E[z_9(n) | x_w(\overline{n}_w(n))] \\
&= \sum_{w \in W} < \eta_w(n), \tilde{x}^*(n+1) - x_w(\overline{n}_w(n)) > (1 - e^{-\mu_w T(n - \overline{n}_w(n))}) \\
&\leq \sum_{w \in W} \|\eta_w(n)\| [\|\tilde{x}_w^*(n+1) - x_w(\overline{n}_w(n))\| + \|s_w(\overline{n}_w(n))\|](1 - e^{-MT\overline{n}}).
\end{aligned}
$$

By the Holder's inequality, (6), and (4.1),

$$
\begin{aligned}
E[\|\eta_w(n)\| \|\tilde{x}_w^*(n+1) - x_w(\overline{n}_w(n))\|] &\leq \left(E\|\eta_w(n)\|^2\right)^{\frac{1}{2}} \left(E\|\tilde{x}_w^*(n+1) - x_w(\overline{n}_w(n))\|^2\right)^{\frac{1}{2}} \\
&\leq A_{27} (A_5 \varepsilon + A_6 e^{-2\mu n T})^{\frac{1}{2}}.
\end{aligned}
$$

By the Holder's inequality again

$$
E[\|\eta_w(n)\| \|s_w(\overline{n}_w(n))\|] \leq A_{27} \left(E\|s_w(\overline{n}_w(n))\|^2\right)^{\frac{1}{2}}.
$$

The above two inequalities imply

$$
E[z_9(n)] \leq A_{28} (A_5 \varepsilon + A_6 e^{-2\mu n T})^{\frac{1}{2}} + A_{28} \sum_{w \in W} \left(E\|s_w(\overline{n}_w(n))\|^2\right)^{\frac{1}{2}}. \qquad (19)
$$

Finally, by combining inequalities (12)-(18), we get

$$E[D(n+1) \leq E[D(n)] + \left(1-e^{-\mu T}\right) e^{-MT\overline{n}}[D^* - E(D(n))] + b(n,\varepsilon),$$

where

$$\begin{aligned} b(n,\varepsilon) = {} & A_{29}\varepsilon + A_{31}\left(A_5\varepsilon + A_6 e^{-2\mu n T}\right)^{\frac{1}{2}} + A_{34}\sum_{w\in W}\left(A_5\varepsilon + A_6 e^{-2\mu \overline{n}_w(n)T}\right)^{\frac{1}{2}} \\ & + A_{32}\left(A_3 E\|s_w(\overline{n}_w(n))\|^2 + A_2\varepsilon + A_4 e^{-2\mu n T}\right)^{\frac{1}{2}} \\ & + A_{33}\sum_{m=0}^{C}\sum_{w\in W} E\|s_w(\overline{n}_w(n-m))\|^2 + \left(\frac{A_{15}}{\alpha} + A_{28}\right)\sum_{w\in W}\left(E\|s_w(\overline{n}_w(n))\|^2\right)^{\frac{1}{2}}. \end{aligned}$$

Hence,

$$E[D(n+1)] - D^* \leq \left(1 - e^{-MT\overline{n}}\left(1-e^{-\mu T}\right)\right)[E[D(n)] - D^*] + b(n,\varepsilon).$$

Repeated application of the above inequality yields

$$E[D(n)] - D^* \leq \beta^n(\overline{n})[D(0) - D^*] + a(n,\varepsilon),$$

where

$$a(n,\varepsilon) = \sum_{i=1}^{n-1}\beta^{n-i-1}(\overline{n}) b(i,\varepsilon).$$

Since $b(n,\varepsilon)$ is bounded,

$$\overline{\lim_{n\to\infty}} a(n,\varepsilon)$$

exits. Thus completes the proof of (7).

To show (a) we have to bound $z_4(n)$ differently. By Lemma 5.1, we have

$$z_4(n) \leq \left(1-e^{-\mu T}\right) e^{-MT\overline{n}}\frac{\delta}{\alpha}\sum_{w\in W}\|s_w(\overline{n}_w(n))\|^2. \tag{20}$$

Using (20), (14), (16), (18) to bound $E[z_4(n)]$, $E[z_5(n)]$, $E[z_2(n)]$, $E[z_3(n)]$, we get

$$\begin{aligned} E[D(n+1)] \leq {} & E[D(n)] - \frac{A_{35}}{\alpha}E\|\overline{s}(n)\|^2 + A_{33}\sum_{m=0}^{C} E\|\overline{s}(n-m)\|^2 \\ & + A_{29}\varepsilon + A_{30}e^{-2\mu n T} + A_{17}(A_5\varepsilon + A_6 e^{-2\mu n T})^{\frac{1}{2}}, \end{aligned} \tag{21}$$

where

$$\bar{s}(n) = [s_w(\bar{n}_w(n))]_{w \in W} \,.$$

Choose α small enough so that

$$A(\alpha) = \left(\frac{A_{35}}{\alpha(C+1)} - A_{33} \right) > 0.$$

Now by the convexity of D,

$$\begin{aligned} D(n) - D^* \leq & \sum_{w \in W} < \eta_w(n), x_w(n) - x_w^* > \\ = & \sum_{w \in W} \{ < d_w(n) - \eta_w(n), x_w(n) - x_w^* > + < \eta_w(n), x_w(n) - x_w^*(n+1) > \\ & + < \eta_w(n), x_w^*(n+1) - \bar{x}_w(n) > + < \eta_w(n), \bar{x}_w(n) - x_w^* > . \end{aligned} \tag{22}$$

By Lemma 5.4,

$$\begin{aligned} & \sum_{w \in W} < d_w(n) - \eta_w(n), x_w(n) - x_w^* > \\ & \leq A_{36} \sum_{m=1}^{C} \|x(n-m+1) - x(n-m)\| \sum_{w \in W} \|x_w(n) - x_w^*\| \\ & \leq A_{37} \sum_{m=1}^{C} \|x(n-m+1) - x(n-m)\| \|x(n) - x^*\|. \end{aligned}$$

Hence,

$$\begin{aligned} & E[\sum_{w \in W} < d_w(n) - \eta_w(n), x_w(n) - x_w^* >] \\ \leq & A_{38} \sum_{m=1}^{C} \left(E\|x(n-m+1) - x(n-m)\|^2\right)^{\frac{1}{2}} \\ \leq & A_{39} \sum_{m=1}^{C} \left[E\|\bar{s}(n-m)\|^2 + A_{40}\varepsilon + A_{41}e^{-2\mu T n}\right]^{\frac{1}{2}} \\ \leq & A_{39} \sum_{m=1}^{C} \left(E\|\bar{s}(n-m)\|^2\right)^{\frac{1}{2}} + A_{42}\left(A_{40}\varepsilon + A_{41}e^{-2\mu T n}\right)^{\frac{1}{2}} . \end{aligned} \tag{23}$$

Now we bound the second term of (20),

$$\begin{aligned}
&E[< \eta_w(n), x_w(n) - x^*_w(n+1) > | x_w(\overline{n}_w(n))] \\
&= < \eta_w(n), x^*_w(n+1) - x_w(n+1) > \\
&\quad + < \eta_w(n), x_w(\overline{n}_w(n)) - \tilde{x}^*_w(\overline{n}_w(n)+1) > e^{-\mu_w T(n-\overline{n}_w(n))} \\
&= < \eta_w(n), x^*_w(n+1) - x_w(n+1) > (1 - e^{-\mu_w T(n-\overline{n}_w(n))}) \\
&\quad - < \eta_w(n), s_w(\overline{n}_w(n)) > e^{-\mu_w T(n-\overline{n}_w(n))}.
\end{aligned}$$

Hence,

$$\begin{aligned}
&E\left[\sum_{w\in W} < \eta_w(n), x_w(n) - x^*_w(n+1) >\right] \\
&\leq A_{43}(A_5\varepsilon + A_6 e^{-2\mu n T})^{\frac{1}{2}} + A_{44}(E\|\overline{s}(n)\|^2)^{\frac{1}{2}}.
\end{aligned} \tag{24}$$

Also,

$$\begin{aligned}
&E\left[\sum_{w\in W} < \eta_w(n), x^*_w(n+1) - \|x(n+1) - x(n)\|(n) >\right] \\
&\leq E\left[\sum_{w\in W} \frac{A_1}{\alpha} r_w(\overline{n}_w(n))\|s_w(\overline{n}_w(n))\|\right] \\
&\leq A_{45}\left(E\|\overline{s}(n)\|^2\right)^{\frac{1}{2}}.
\end{aligned} \tag{25}$$

By a similar argument, the fourth term in (22) is bounded by

$$E\left[\sum_{w\in W} < \eta_w(n), x^*_w(n) - x^*_w >\right] \leq A_{46}(A_5\varepsilon + A_6 e^{-2\mu n T})^{\frac{1}{2}}. \tag{26}$$

Now (22)-(26) imply

$$\begin{aligned}
E[D(n)] - D^* &\leq A_{39}\sum_{m=1}^{C}(E\|\overline{s}(n-m)\|^2)^{\frac{1}{2}} \\
&\quad + A_{47}(A_5\varepsilon + A_6 e^{-2\mu n T})^{\frac{1}{2}} + A_{44}(E\|\overline{s}(n)\|^2)^{\frac{1}{2}}.
\end{aligned}$$

Thus there exist positive constants A_{48}, A_{49} and $n_1 > 0$, such that $\forall n \geq n_1$,

$$E[D(n)] - D^* \leq A_{48}\sum_{m=1}^{C}(E\|\overline{s}(n-m)\|^2)^{\frac{1}{2}} + A_{49}\varepsilon^{\frac{1}{2}}. \tag{27}$$

Now there exists a positive constant A_{50} such that we can rewrite (27) into

$$A_{50}[E(D(n)) - D^*] - A_{51}\varepsilon^{\frac{1}{2}} \le \left[A(\alpha)\sum_{m=0}^{C} E\|\bar{s}(n-m)\|^2\right]^{\frac{1}{2}}. \tag{28}$$

Now using (21), there exists $n_2 \ge n_1$ such that for all $n \ge n_2$, $\forall \varepsilon \le 1$,

$$E[D(n+1)] \le E[D(n)] - \frac{A_{35}}{\alpha} E\|\bar{s}(n)\|^2 - A_{33}E\|\bar{s}(n-m)\|^2 + A_{52}\varepsilon^{\frac{1}{2}}. \tag{29}$$

Repeating (29) we get

$$\begin{aligned} E[D(n)] &\le D(0) + \sum_{i=1}^{n-1}\sum_{m=0}^{C}\left[-\left(\frac{A_{34}}{\alpha(C+1)} + A_{33}\right)\right] E\|\bar{s}(i-m)\|^2 + \frac{A_{53}}{C+1}\varepsilon^{\frac{1}{2}} \\ &= D(0) + \sum_{i=1}^{n-1} g(i), \end{aligned} \tag{30}$$

where

$$g(i) = \sum_{m=0}^{C} -A(\alpha)E\|\bar{s}(i-m)\|^2 + \frac{A_{53}}{C+1}\varepsilon^{\frac{1}{2}}.$$

Let

$$A_{53} = (A_{52})^{\frac{1}{2}}.$$

Now for all $n \ge n_2$, $\varepsilon \le 1$, if

$$E[D(n)] - D^* \ge (A_{50})^{-1}[(A_{51}+1)\varepsilon^{\frac{1}{2}} + A_{53}\varepsilon^{\frac{1}{4}}],$$

then

$$A_{50}(E[D(n)] - D^*) - A_{51}\varepsilon^{\frac{1}{2}} \ge \varepsilon^{\frac{1}{2}} + A_{53}\varepsilon^{\frac{1}{4}}.$$

By (28)

$$A(\alpha)\sum_{m=0}^{C} E\|\bar{s}(n-m)\|^2 \ge (\varepsilon^{\frac{1}{2}} + A_{53}\varepsilon^{\frac{1}{4}})^2 \ge \varepsilon + A_{52}\varepsilon^{\frac{1}{2}},$$

which implies

$$g(n) \le -\varepsilon.$$

Thus

$$\overline{\lim_{n\to\infty}} E[D(n)] - D^* \le (A_{50})^{-1}[(A_{51}+1)\varepsilon^{\frac{1}{2}} + A_{53}\varepsilon^{\frac{1}{4}}],$$

which is (8). Now (9), (10), and (11) can be proved by using the same argument as that of Theorem 4.5. *Q.E.D.*

CHAPTER SIX
THE PATH ASSIGNMENT PROBLEM

6.1 Introduction

The VC path assignment problem and gradient projection routing with metered path assignment are studied in this chapter.

By nature, the VC path assignment problem is a dynamic routing problem in the sense that the assignment (routing) is based on the instantaneous state of the network. More precisely, the paths are assigned on the basis of the routing variables determined at the latest routing updates, i.e., the desired flows, and the instantaneous number of VCs on each path. The problem is formulated as an optimal queueing control problem in section 6.2. Dynamic routing problems are generally known to be difficult [Ephremides et.al. 1978].

We study the path assignment for two purposes. First, we need to design a *best* assignment policy so that gradient projection routing can perform well. Second, we need to analytically compute or estimate the statistics of the path flows resulting from such a policy. These statistics are crucial in the convergence proof since they determine the transient behavior of the path flows during a routing interval.

The sum of the square of the difference between the actual and the desired VC numbers on each path at the end of a routing interval, is a natural cost for the queueing control problem. Intuition suggests that an optimal control should take the VC departure and arrival rates into account. Although we have not found an optimal control, we have found that a control based on the deficiency in the desired numbers of VC is good in some sense. In fact we analytically estimate the quadratic cost resulting from our metering rule based on the deficiency. At steady-state, we show that this rule incurs a smaller cost than the randomized policy.

With the estimates of the quadratic cost incurred by the metering rule, we upper bound the first and the second order statistics of the transient flows resulting from a routing update. With these bounds, we show a similar convergence result of the gradient projection algorithm with the simplex constraint, except with an additional condition. The added condition is that, either the scaling matrix $\hat{M}_w(n)$ is a scalar multiple of an identity matrix, or the routing interval must be sufficiently *long*. We believe that this added condition is unnecessary; it arises because we are only using bounds on the first and the second order statistics on the transient path flows. Should these statistics or tighter bounds be analytically available, the added condition would be discarded.

6.2 The Path Assignment Problem

We first motivate the path assignment problem by considering the first step in the convergence proof of Theorem 4.1:

$$\begin{aligned} D(n+1) \leq & D(n) + \sum_{w \in W} < d_w(n), x_w(n+1) - x_w(n) > \\ & + \frac{\mathrm{L}}{2}\|x(n+1) - x(n)\|^2, \\ \leq & D(n) + \sum_{w \in W} < d_w(n), x_w(n+1) - x_w(n) > \\ & + \mathrm{L}\{\|x(n+1) - x^*(n+1)\|^2 + \|x^*(n+1) - x(n)\|^2\}. \end{aligned}$$

We need to compute analytically the expectation of the above first and second order terms of $x(n+1) - x(n)$. Since $x_p^*(n+1)$ is proportional to $N_p^*(n+1), \forall p \in P_w$, we would like to bound the following quadratic term,

$$E[\sum_{p \in P_w} (N_p(n+1) - N_p^*(n+1))^2],$$

which shall become the cost for the path assignment problem. Once this quadratic cost is found, we can use the Hölder's inequality to compute the first order term, which is shown in lemma 6.4.

We now reformulate the problem, for any OD pair, as an optimal queueing control problem. Consider a queueing system with l queues and each queue having an infinite number of exponential servers with service rate μ. Assume that the desired numbers of customers $N_k^*, k = 1, 2, \ldots, l$ are given and known to the controller. The customers arrive at a Poisson rate λ, and each customer is to be assigned by the controller to be served at one of the queues, upon arrival. At time t, the controller observes the system state, i.e., the number of customers at each queue, and make the control decision, $u_k(t)$, the probability of assigning the customer arriving at t to queue k. The controller does not have the knowledge of either λ or μ. The objective is to minimize the cost:

$$E^u[\sum_{k=1}^{l}(N_k(T) - N_k^*)^2],$$

where T is the terminal time, and $E^u[.]$ is the expectation conditioned on the control $u(t)$, and the system is started at time 0 with arbitrary initial conditions.

First we recall that Ephremides, Varaiya, and Walrand [1980] solved a similar problem. They considered two queues, with each queue having an identical exponential server. They proved that sending the arriving customers to the shortest queue optimizes a certain cost. As our problem is quite similar to theirs, a natural guess of the optimal control would be sending the arriving customers to the relatively shortest queue, with relative queue size measured by

$$N_k(t) - N_k^*.$$

We shall call this rule the *deficiency* rule.

This rule is in fact very natural. Consider the case where a customer at t is to be added to the system, the immediate resulting cost is minimized if the customer is sent to the queue with the most negative $N_k(t) - N_k^*$. Alternately the marginal cost of adding a customer to queue k at time t is precisely $2(N_k(t) - N_k^*) + 1$. This observation suggests the above deficiency rule.

Is this deficiency rule optimal in minimizing the quadratic cost? We doubt it is. Consider the following scenario: two queues with $N_1(0) = 999$, $N_2(0) = 1$, $N_1^* = 1000$, $N_2^* = 2$, and there is only one arrival followed by one departure to occur between $[0, T]$. Clearly, the arriving customer can be sent to either one of the queues, according to the deficiency rule. However, the cost incurred due to the assignment to queue 1 is about 2 whereas the cost incurred by the assignment to queue 2 is about 4. In the calculation we have assumed that the departure at a queue occurs with a probability proportional to the queue size. The problem with the deficiency rule is that it does not take into account the different departure rates.

The above scenario suggests that an optimal control should estimate μ or even λ in an effort to base its control on the expected queue length at the next arrival or the terminal time T. However, the result of Ephremides et. al. [1980] suggests that an assignment policy based on expected queue lengths can often be non-optimal. Thus, we believe that an optimal control must have a different form than what we have considered so far. However, in practice, the metering rule should be simply implementable. The deficiency rule certainly meets this requirement.

We need additional notations to present the bounding result. Let $E[.]$ represent expectation conditioned on the deficiency rule for the subsequent development. Let $V(t)$ be the quadratic cost incurred at time t:

$$V(t) = \sum_{k=1}^{l} E[(N_k(t) - N_k^*)^2].$$

Let $u(t)$ be the deficiency rule. Let N be the initial total number of customers in the system satisfying the equation:

$$N = \sum_{k=1}^{l} N_k^*.$$

We denote the queue size vector at time t by $N(t)$. Let the long-term average number of customers be $\tilde{N}$:

$$\tilde{N} = \frac{\lambda}{\mu},$$

which comes from the Little's rule. Define the normalized desired number of customers to be $\tilde{N}_k^*$:

$$\tilde{N}_k^* = \frac{N_k \tilde{N}}{N}.$$

In most of the equations presented in the rest of this section, the equality or inequality holds true almost surely. However, for simplicity we omit the phrase *a.s.* in these equations.

The approach we take is to find a first order differential inequality which $V(t)$ satisfies. The trick is to use the first order Taylor series expansion on the conditional expectations.

Theorem 6.1

$$\frac{d}{dt}V(t) \leq -2\mu V(t) + 2\mu E[\sum_{k=1}^{l} N_k(t)]|N - \tilde{N}| + \lambda.$$

Proof: Fix any $0 \leq t \leq T$, consider any $0 \leq h \leq T - t$ small enough,

$$V(t) = E[E[\sum_{k=1}^{l}(N_k(t+h) - N_k^*)^2 | N(t)]].$$

Expand the conditional expectation in the above equation by the Taylor series expansion:

$$\begin{aligned} &E[\sum_{k=1}^{l}(N_k(t+h) - N_k^*)^2 | N(t)] \\ &= \sum_{k=1}^{l}\{(N_k(t) - N_k^*)^2(1 - h\mu N_k(t) - h\lambda u_k(t)) \\ &+ (N_k(t) - N_k^* - 1)^2 h\mu N_k(t) + (N_k(t) - N_k^* + 1)^2 h\lambda u_k(t)\} + o(h^2). \end{aligned}$$

Simple algebra leads to

$$\begin{aligned} &E[V(t+h)|N(t)] \\ &= \sum_{k=1}^{l}\{(N_k(t) - N_k^*)[(N_k(t) - N_k^*) - 2\mu N_k(t)h + 2\lambda u_k(t)h]\} \\ &+ h\mu E[\sum_{k=1}^{l} N_k(t)] + h\lambda + o(h^2). \end{aligned}$$

We can factor out $V(t)$ in the above equation to get

$$\begin{aligned} &E[V(t+h)|N(t)] \\ =&E\{\sum_{k=1}^{l}(1-2h\mu)(N_k(t)-N_k^*)^2 \\ &+2h\mu(N_k(t)-N_k^*)(\tilde{N}u_k(t)-N_k^*)\} \\ &+h\mu E[\sum_{k=1}^{l}N_k(t)]+h\lambda+o(h^2). \end{aligned}$$

Hence

$$\begin{aligned} V(t+h) = &(1-2h\mu)V(t)+2h\mu E[E[z_1(t)+z_2(t)|N(t)]] \\ &+h\lambda E[\sum_{k=1}^{l}N_k(t)]+o(h^2), \end{aligned}$$

with

$$z_1(t)=\tilde{N}\sum_{k=1}^{l}(N_k(t)-N_k^*)(u_k(t)-\frac{N_k^*}{N}), \qquad (1)$$

$$z_2(t)=\sum_{k=1}^{l}(N_k(t)-N_k^*)(\tilde{N}_k^*-N_k^*). \qquad (2)$$

We now bound $z_1(t)$ and $z_2(t)$. From (1) it is clear, due to the fact that $\frac{N_k^*}{N}$ are non-negative numbers summing to unity, that if $u_j(t)=1$ for some j satisfying

$$j=\arg\min_{1\le k\le l}[N_k(t)-N_k^*],$$

then $z_1(t)\le 0$. To bound $z_2(t)$, note that

$$\begin{aligned} z_2(t) &= (\tilde{N}-N)\sum_{k=1}^{l}(N_k(t)-N_k^*)\frac{N_k^*}{N} \\ &\le E[\sum_{k=1}^{l}N_k(t)]|N-\tilde{N}|. \end{aligned}$$

Hence the result follows by taking the limit $h\downarrow 0$. *Q.E.D.*

An alternate metering rule used in practice is the following rule (referred to as the *fraction* rule): a customer arriving at time t is assigned to queue i where

$$i=\arg\min_{k=1,2,\ldots,l}\left\{\frac{N_k(t)}{N_k^*}\right\}.$$

Note that this rule does not minimize the term $z_1(t)$ in (1). The deficiency rule is *optimal* in the sense that it minimizes $z_1(t)$.

The above result has an interesting corollary when $\tilde{N} = N$. The corollary says that if the queueing system has reached a steady-state, then metering is better than randomization.

Corollary 6.2

Let

$$V_1(t) = E[\sum_{k=1}^{l}(N_k(t) - N_k^*)^2 | u(t) = \text{the deficiency rule}],$$

$$V_2(t) = E[\sum_{k=1}^{l}(N_k(t) - N_k^*)^2 | u(t) = \text{randomization}],$$

$$e(t) = V_1(t) - V_2(t).$$

Suppose $\tilde{N} = N$, then $e(t)$ satisfies the differential inequality:

$$\frac{d}{dt}e(t) \leq -2\mu e(t).$$

The proof of the above corollary is omitted as it can be easily derived using a similar argument to that of Theorem 6.1. Assume the condition of the above corollary and that $e(0) = 0$, then it is clear that $e(t) \leq 0$, for all $t \geq 0$, i.e., the deficiency rule incurs a smaller cost than the randomization rule for all time.

6.3 The Convergence Result With Metered Path Assignment

By studying the proofs of the convergence results (Theorems 4.5 and 5.5), we note that there are two crucial steps. First, the first order terms must constitute a negative quadratic term of $s(n)$, and be inversely proportional to the stepsize α. Second, the second order terms must be proportional to a quadratic form of $s(n)$ plus terms which decay to zero as $\varepsilon \to 0$ and $n \to \infty$. Thus we only need to show modified versions of lemmas 4.2 and 4.3. We will omit the asynchronous case since the lemmas and proofs are more complicated, yet provide little additional insight. The reader can easily generalize the lemmas 6.3 and 6.4 in this section, and follows the spirit of the proof of Theorem 5.5 to obtain the convergence result for the asynchronous case.

With the result of Theorem 6.1, we show the bounds for the second order terms in the following lemma.

Lemma 6.3 (c.f. Lemma 4.2) There exist positive constants $A_1 - A_3$ such that

$$E[\|x(n+1) - x(n)\|^2 | x(n)] \leq A_1\varepsilon + A_2 E[\|s(n)\|^2] + A_3 |r_w(n) - \bar{r}_w|.$$

Proof: By the inequality $(a+b)^2 \leq 2(a^2+b^2)$ we have

$$E[\|x(n+1) - x(n)\|^2 | x(n)] \leq 2E[\|x(n+1) - x^*(n+1)\|^2 | x(n)] + 2\|s(n)\|^2.$$

Fix any w, by Theorem 6.1 and the variation of constants formula (e.g., (A9) of [GB]),

$$E[\|x_w(n+1) - x_w^*(n+1)\|^2 | x(n)] \leq e^{-2\mu_w T}\|s_w(n)\|^2 + A_4(1 - e^{-\mu_w T})|r_w(n) - \bar{r}_w|$$
$$+ \varepsilon \overline{\gamma}_w \left(1 - e^{-\mu T}\right)\left(\bar{r}_w + e^{-\mu_w T} r_w(n)\right). \qquad (3)$$

Now the result follows from the above inequality. *Q.E.D.*

The next lemma is a combination of the modified lemmas 4.3 and 4.4, and shows the sufficient decrease in the cost due to the first order terms. The algorithm assumed here is the gradient projection with simplex constraints (equation (1) of chapter 4).

Lemma 6.4 There exist positive constants $A_5 - A_6$ such that $\forall\, w \in W$

$$E[< d_w(n), x_w(n+1) - x_w(n) >] \leq -\left(\frac{\delta}{\alpha} - \frac{\Delta}{\alpha} e^{-\mu_w T}\right) E\|s_w(n)\|^2$$
$$+ A_5 \varepsilon^{\frac{1}{2}} + A_6 E\left[|r_w(n) - \bar{r}_w|^{\frac{1}{2}}\right].$$

Proof: By adding and subtracting terms, we get

$$< d_w(n), x_w(n+1) - x_w(n) >= z_1(n) + z_2(n),$$

with

$$z_1(n) =< d_w(n), x_w^*(n+1) - x_w(n) >,$$

$$z_2(n) =< d_w(n), x_w(n+1) - x_w^*(n+1) > .$$

By Lemma 4.3 we have

$$z_1(n) \leq -\frac{\delta}{\alpha}\|s_w(n)\|^2. \tag{4}$$

By Lemma 4.4 and equation (16), we have, with $y = x_w(n)$,

$$z_2(n) \leq \frac{\Delta}{\alpha}\|s_w(n)\|\|x_w^*(n+1) - x_w(n)\|.$$

By the Holder's inequality,

$$E[z_2(n)] \leq \frac{\Delta}{\alpha}(E[\|s_w(n)\|^2])^{\frac{1}{2}}(E[\|x_w^*(n+1) - x_w(n)\|^2)^{\frac{1}{2}}.$$

By equation (3) (in the proof of lemma 6.3), there exist positive constants A_5 and A_6 such that

$$E[z_2(n)] \leq \frac{\Delta}{\alpha}e^{-\mu_w T}E\|s_w(n)\|^2 + A_5\varepsilon^{\frac{1}{2}} + A_6 E\left[|r_w(n) - \bar{r}_w|^{\frac{1}{2}}\right]. \tag{5}$$

The result follows from (4) and (5). *Q.E.D.*

From the above lemma, in order to guarantee the convergence, it is crucial that $A(\alpha)$:

$$A(\alpha) = \frac{\delta}{\alpha} - \frac{\Delta}{\alpha}e^{-\mu_w T}, \tag{6}$$

is positive. $A(\alpha)$ is positive if either $\delta = \Delta$ or $T\mu_w$ is large enough. However, we believe that such conditions should not be necessary. It is due to the weakness of our bounding result. The next lemma is the counterpart of Lemma 6.4. The algorithm assumed for Lemma 6.4.a is the gradient projection with the orthant constraint. The proof is similar in the spirit to that of Lemma 6.4.

Lemma 6.4.a There exist positive constants $A_7 - A_{10}$ such that $\forall w \in W$

$$E[< d_w(n), x_w(n+1) - x_w(n) >]$$
$$\leq -\left(\frac{A_7}{\alpha} - \frac{A_8}{\alpha}e^{-\mu_w T}\right) E\|s_w(n)\|^2 + A_9 \varepsilon^{\frac{1}{2}} + A_{10} E\left[|r_w(n) - \bar{r}_w|^{\frac{1}{2}}\right].$$

Proof: First write

$$< d_w(n), x_w(n+1) - x_w(n) >= z_1(n) + z_2(n),$$

with

$$z_1(n) =< d_w(n), x_w^*(n+1) - x_w(n) >,$$

$$z_2(n) =< d_w(n), x_w(n+1) - x_w^*(n+1) > .$$

By Lemma 4.3.a, we have

$$z_1(n) \leq -\frac{A_7}{\alpha}\|s_w(n)\|^2. \tag{7}$$

By Lemma 4.4.a, we have, with $y = x_w(n)$,

$$z_2(n) \leq \frac{A_8}{\alpha}\|s_w(n)\|\|x_w^*(n+1) - x_w(n)\|.$$

By the Holder's inequality,

$$E[z_2(n)] \leq \frac{A_8}{\alpha}(E[\|s_w(n)\|^2])^{\frac{1}{2}}(E[\|x_w^*(n+1) - x_w(n)\|^2)^{\frac{1}{2}}.$$

By equation (3) (in the proof of lemma 6.3), there exist positive constants A_9 and A_{10} such that

$$E[z_2(n)] \leq \frac{A_8}{\alpha}e^{-\mu_w T}E\|s_w(n)\|^2 + A_9\varepsilon^{\frac{1}{2}} + A_{10}E\left[|r_w(n) - \bar{r}_w|^{\frac{1}{2}}\right]. \tag{8}$$

The result follows from (7) and (8). *Q.E.D.*

Finally, we present the corresponding convergence result of the synchronous gradient projection algorithm with the deficiency rule and the simplex constraints. It is identical with its counterpart which uses the randomized path assignment, except the above mentioned extra condition (6).

Theorem 6.5 There exist a positive constants, c_1, and a positive function $a(n, \varepsilon)$ (which depends only on the system parameter set $\mathbf{A}$ and α), such that $\forall n = 0, 1, 2, \ldots$

$$-c_1 e^{-\mu n T} \leq E[D(n)] - D^* \leq e^{-\mu n T}[D(0) - D^*] + a(n, \varepsilon), \tag{9.1}$$

$$\overline{\lim_{n\to\infty}}\, a(n, \varepsilon) < \infty. \tag{9.2}$$

Furthermore, if $\delta = \Delta$ or if

$$\frac{\delta}{\alpha} - \frac{\Delta}{\alpha} e^{-\mu T} > 0,$$

then there exists a scalar $\overline{\alpha} > 0$, such that $\forall \alpha \in (0, \overline{\alpha}], \forall \varepsilon \leq 1$, the following is true:

(a)There exists a positive constant c_2 (which depends on the system parameter set $\mathbf{A}$ and α) such that

$$\overline{\lim_{n\to\infty}}\, E[D(n)] - D^* \leq c_2 \varepsilon^{\frac{1}{4}}, \tag{10}$$

$$\lim_{\varepsilon\to 0} \overline{\lim_{n\to\infty}}\, E\|s(n)\|^2 = 0, \tag{11}$$

$$\lim_{\varepsilon\to 0} \overline{\lim_{n\to\infty}}\, a(n, \varepsilon) = 0, \tag{12}$$

$$\lim_{\varepsilon\to 0} \overline{\lim_{n\to\infty}}\, E[D(n)] = D^*. \tag{13}$$

(b) If, in addition, $\overline{D}_{ij}$ is strictly convex for all $(i, j) \in L$, then

$$\lim_{\varepsilon\to 0} \overline{\lim_{n\to\infty}}\, E\|F(n) - F^*\|^2 = 0.$$

The proof of the above Theorem is very similar to that of Theorem 4.5. The first half is almost identically the same. The major difference is in proving (a), where the counterpart of inequality (48.1) needs the lemma 6.4. We shall omit the proof.

We also omit here the corresponding convergence result of asynchronous gradient projection routing with metered path assignment. The statement of the corresponding theorem and the corresponding proof is similar to that of their counterparts in chapter 5. Again the major difference is the counterpart of the condition (6) and the lemmas and proofs associated with it.

CHAPTER SEVEN
ROUTING SIMULATION

7.1 Introduction

Real world data network routing is so complicated that no analytical model is capable of capturing all its characteristics. We present in this chapter a digital simulation which accounts for the many details of practical routing omitted by the analytical models presented so far.

The purpose of the simulation is to compare qualitatively the analytical results with the simulated performance and to supplement the analytical study with additional insight. Since our study is not an extensive Monte Carlo simulation, we shall mostly extract qualitative observations instead of drawing quantitative conclusions.

In particular there are many factors that affect the performance of virtual circuit routing. It is thus interesting to compare routing performances due to variations of these factors. First, there are the factors and parameters associated with the routing algorithms and routing update implementation : stepsize choice, algorithmic variations, path assignment policy, and the policies associated with flow measurement and the flooding of update packets. Second, different traffic conditions also produce different routing performance. Lastly, there are the parameters associated with the load : VC arrival rates, VC departure rates, packet arrival rates per VC, and the average packet size.

The simulation software is written as an extension of a general purpose network simulation system – OPNET – developed at MIT by Cohen and Baraniuk [1986], on a Data General MV10000, a 32-bit super-mini. The software models routing with many details : packets, virtual circuits, and routing implementations.

We choose a test network, Net8, from Bertsekas, Gafni, and Vastola [1978] for detailed simulation. Net8 (see Figure 7.2) is an 8-node network with good connectivity.

This chapter is organized as follows. The digital routing model is described in some details in section 7.2. A discussion on the simulation parameters is given in section 7.3. Section 7.4 introduces the test network and section 7.5 contains the main simulation results and interpretations. The graphs of the simulation runs are provided in Appendix A, while the codes are given in Appendix B.

7.2 Simulation Model

We first introduce OPNET, the underlying package upon which the simulation program is written. OPNET is a dynamic stochastic hierarchical general purpose network simulation system. The system is discrete-time in the sense that all the finest-grain events are associated with a single time interval and the time is advanced by fixed increments. The smallest fixed increment is referred to as a simulation cycle.

The simulation model can be roughly categorized into three levels : packet level, VC level, and routing level, which are now described below.

Packet Level

The simulator actually generates packets, both data and update (control) packets, which will traverse the network and eventually be read and destroyed upon exiting the network. The arrival of the packets for each VC is either an approximate Poisson processes or a deterministic process. Packet length (in bits) is either constant or approximately exponentially distributed. The packets are capable of carrying such information as the flow rate of a certain link at a certain time.

The code simulates seven components which deal with packets : load generators, routing processors, queues, transmitters, receivers, transmission lines, and pipes. Their functions are described below.

Load Generator : generates packets as specified.

Routing Processor : routes the transit packets, destroys/reads packets destinated for the local node, and forwards packets originated locally.

Queue : stores packets temporarily, with a non-preemptive fixed priority queueing discipline in which the update packets have the higher priority.

Transmitter : repeatedly takes the top packet from the queue attached at its input port and transmits the packet (over a transmission line attached). To model continuous transmission, if the length of the current packet to be sent plus the total accumulated packet length transmitted in the current cycle is greater than the pre-specified rate (in bits/cycle), the transmitter will break the packet into two smaller packets. The first small packet is of the length equal to the remaining bandwidth of the transmission line at the current cycle, and the second small packet takes the remaining bitsize of the original packet. The first small packet will be transmitted in the current cycle while the second small packet will be the first packet to be sent in the next cycle.

Transmission Line : forwards, without error (no ARQ), all the packets transmitted in a cycle by the transmitter at one end to the receiver at the other end at the beginning of the next cycle; thus a delay of one cycle is imposed on the packets traversing a transmission line.

Receiver : receives all the packets forwarded by the transmission line attached and sends them to the output port.

Pipe : transports the packets inside a node; the packets are forwarded without any delay.

A schematic diagram for a typical node is depicted below (the controller will be introduced shortly).

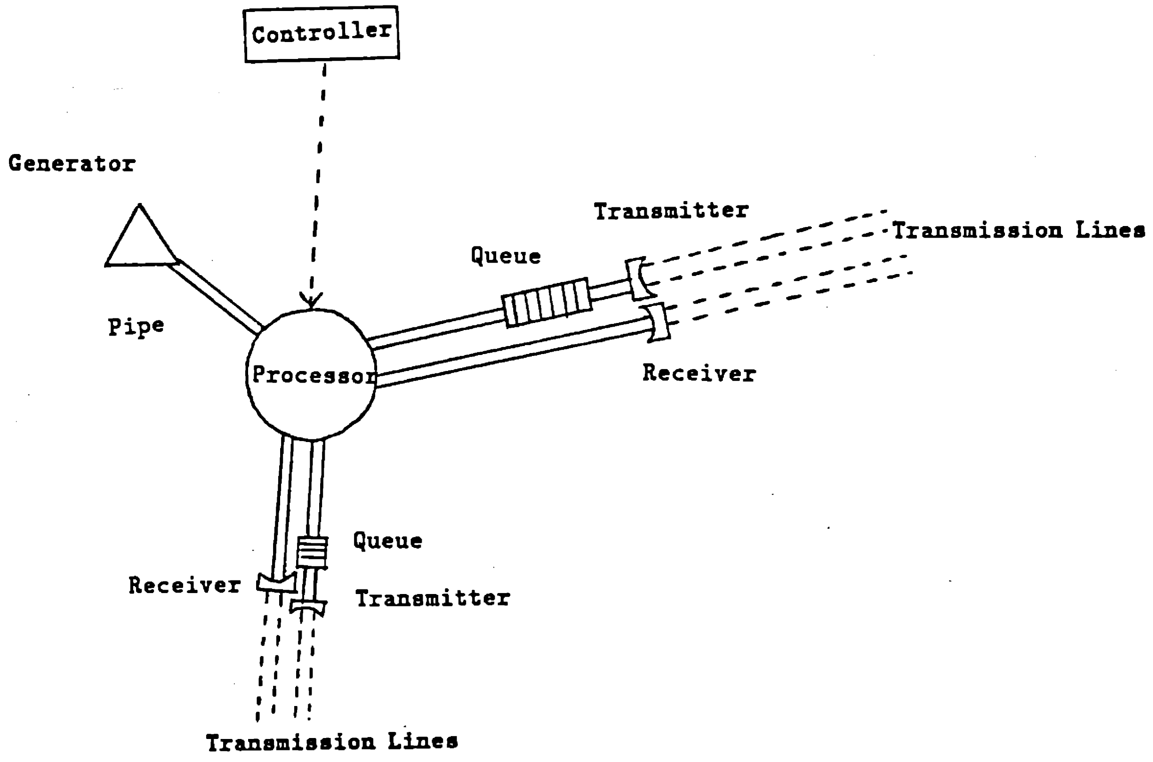


Figure 7.1 A Typical Node

VC Level

For each node, there is a controller which generates and keeps track of the VCs of the OD pairs originating from the node. The VCs are usually generated according to approximate Poisson processes, and their durations are roughly exponentially distributed. To simulate a deterministic routing environment, the program has an option of keeping the total number of VCs constant for each OD pair. Also every packet of each VC originated at a node is generated by the local load generator and is assigned, by the local routing processor, a path number. The packets with a particular path number belong to the VCs routed via that path.

There are four ways to assign the arriving VCs: the deficiency rule, the fraction rule, the randomization rule, and the shortest-path rule. The deficiency rule and the fraction rule are described in chapter 6, while the randomization rule is described in chapter 4. The shortest path rule is the path assignment rule for the shortest path routing algorithm described in chapter 3.

Routing Level

The tasks at this level include : flow measurement, communication of update packets, and routing updates. These tasks are carried out jointly by the controllers and the routing processors.

For each directed link, the flow rate (in bits/cycle) is calculated continually as a moving-average of the most recent instantaneous flow rates over a fixed measurement period, at the starting and the ending nodes of the link. For example, if the measurement period is 100 cycles, the flow (or more precisely the short-term average flow rate) at time t, $F_{ij}(t)$ for each link (i,j), which originates at node i and ends at node j, is computed as the average of $f_{ij}(t-1),\cdots,f_{ij}(t-100)$, where $f_{ij}(t)$ is the instantaneous flow rate at time t. The calculation is done both at nodes i and j.

The locally computed short-term flows have to be sent to those nodes which do not have but need this information. By default these flows are forwarded by a flooding mechanism with time stamps. Each node also keeps a table of all the flows of the network – the current flows for the links locally attached, the most recent updates received via the flooding for all other links. To simulate an ideal routing environment, the program has an option that all the nodes have access to all the current short-term flows network-wide.

Asynchronous Operation

We use a reducing threshold method, following the new ARPANET routing scheme, to determine when to initiate a flooding of a link measurement. Each directed link in the network has a pre-assigned node which is responsible for flooding its flow measurement.

Consider a directed link and the node responsible for flooding its flow measurement. The node keeps track of the latest flooding time, the latest flooded flow measurement, the current threshold for the link. If the difference between the current flow measurement and the latest flooded measurement exceeds the current threshold, then a flooding is initiated at the current cycle. If, however, the difference is less than the threshold, no flooding will be initiated, and the threshold will be decremented by a small fixed amount. To speed up the response of the routing algorithm to over-loading, a flooding will be initiated for a link whose utilization factor exceeds a fixed threshold, say, 88%. However, to avoid over-flooding, we keep a minimum difference between the consecutive flooding times for every directed link.

Consider any node in the network. Upon receiving a new flow measurement, the OD pairs originating at the node will carry out a routing update, using the table of network flows available at the node.

The Cost and Routing Algorithms

The cost $\overline{D}_{ij}$ for each link (i,j) is a modified steady-state average queue size of

$M/M/1$ queues (Kleinrock [1976]) :

$$\overline{D}_{ij}(F_{ij}) = \begin{cases} Q_{ij}(F_{ij}), & \text{if } Fij \leq 0.99C_{ij}, \\ Q_{ij}(0.99C_{ij}) + Q'_{ij}(0.99C_{ij})(F_{ij} - 0.99C_{ij}) & \text{otherwise,} \\ +\frac{1}{2}Q''_{ij}(0.99C_{ij})(F_{ij} - 0.99C_{ij})^2, & \end{cases}$$

where

$$Q_{ij}(F_{ij}) = \frac{F_{ij}}{C_{ij} - F_{ij}},$$

and C_{ij} is the capacity of the link (i,j).

The above cost function is thus well defined for all non-negative flows. The above cost equation can be rewritten in terms of the utilization factors ρ_{ij},

$$\rho_{ij} = \frac{F_{ij}}{C_{ij}}.$$

It is clear that with the above cost, the routing performance would be good virtually for any algorithms if the maximum utilization factor is low, say 40% or below. Thus it is more interesting to simulate the situations in which the maximum utilization factor is above 80%.

Two routing algorithms – the *shortest path algorithm* and the *gradient projection algorithm* with the orthant constraints implemented by the deficiency metering rule – are the focus of the simulation. The update equations for the gradient projection algorithms are taken from Bertsekas [1982d] with a simple modification. The diagonal scaling matrix $\hat{M}_w(n)$ has the second derivatives of the *reduced* cost function at time n. The *reduced* cost at time n is obtained by eliminating the path flows of the shortest paths, for all the OD pairs, in the simplex constraint equations in order to retain only non-negativity constraints (see Bertsekas [1982d]). The update equations are as follow. Consider an OD pair w originating at node k, with the shortest path $\overline{p}_w$ at n, and $n \in T_w$,

$$x_p^*(n+1) = [x_p(n) - \alpha L_p^{-1}(n)(\eta_p(n) - \eta_{\overline{p}}(n))]^+, \quad \forall p \in P_w, p \neq \overline{p}_w, \tag{1}$$

$$L_p(n) = \sum_{(i,j)\in S_p} \frac{d^2\overline{D}_{ij}}{dF_{ij}^2}(\hat{F}_{ij,k}(n)),$$

$$x^*_{\overline{p}_w}(n+1) = r_w(n) - \sum_{p \in P_w \ p \neq \overline{p}_w} x^*_p(n+1),$$

where for each p, S_p is the set of links that belong to either the path p or the shortest path $\overline{p}_w$ but not both, at time n. Another algorithm which is less adaptive is to replace (1) by the following.

$$x^*_p(n+1) = [x^*_p(n) - \alpha L_p^{-1}(n)(\eta_p(n) - \eta_p(n))]^+, \quad \forall p \in P_w, p \neq \overline{p}_w. \tag{2}$$

We shall call this second version algorithm 2, the first version algorithm 1. We will focus on the algorithm 1 with the deficiency metering rule, with the fraction metering rule, with the randomization rule; the algorithm 2 with the deficiency rule; and the shortest path algorithm.

7.3 Simulation Parameters

Since the simulator is highly parameterized, it is easy to generate many different kinds of routing environment to simulate. However, the choices must be limited. The following is a set of parameters consistent with most of the assumptions made in the literature (e.g., Rudin [1976], McQuillan and Walden [1977]). This set of parameters are close to the parameters of the new ARPANET routing algorithm.

1). All lines are full duplex with capacity 50 kbits/sec.

2). Data packets average about 960 bits.

3). Update packets average about 176 bits.

4). The measurement period is 10 seconds.

The following is the set of parameters assumed for all our simulations, and is obtained by scaling the above set.

1). All lines are full duplex with capacity 1 or 0.5 kbits/sec.

2). The length of data packets is exponentially distributed with an average of 10 bits.

3). Update packets are fixed at 2 bits.

4). The measurement period is 10 seconds.

It is clear that there is no essential difference between the two sets. To provide granularity in time, the smallest time unit, cycle, in our simulation is set to 0.1 sec.

Two important parameters to be chosen are the VC arrival rates and departure rates. According to the analytical results, the convergence rate of a routing critically depends on the average VC duration. Also, the larger the VC arrival rate (hence the smaller the VC data rate) the better the *many small users* assumption is. Hence we experiment with these parameters. We discover that the larger the VC arrival rate the smaller the long-term deviation from optimality, and the shorter the VC duration the faster the routing converge, given the same routing algorithm. In our typical simulations, the total number of VCs per OD pair is 300, and this number seems to give enough granularity in path flows for the controllers.

One way to consider these VC rates together is to assess the VC *turnover rate* which can be properly defined as the ratio of the average number of new VC arrivals in one update interval to the long-term average number of VCs. At a statistical steady-state, the average numbers of VC arriving and leaving over a fixed interval should be equal. Thus the VC turnover rate measures how fast the VCs are turning over (arriving and leaving) in one update interval. Thus the VC turnover rate also gives a measure of how much control the routing controllers can have over the VCs on each path. If the VC turnover rate is 100% over an update interval, then the controllers have almost full control over the path flows over one update interval. On the other hand, if the VC turnover rate is only 5%, then the best the controllers can do is to change about 5% of the path flows over one update interval.

For our simulations, we use two VC turnover rates. The first is the fast rate corresponding to about 10% and the second is the slow rate corresponding to about 5%. The typical VC arrival and departure rates are listed below.

- Fast Rate: arrival rate = 3.0/sec, departure rate = 0.01/sec.

- Slow Rate: arrival rate = 1.5/sec, departure rate = 0.005/sec.

Another important parameter is the average number of packets per VC per second. This parameter together with the VC arrival and departure rates determine the average number of packets generated per second in the long run. For the test network, we have 16 OD pairs, each averaging 300 VCs at steady-state, with 0.1 packets per second. Together, the long-term average number of packets generated per second is 480. As we typically simulate for 700-1000 seconds, the average number of packets generated per simulation is about $3.36 \times 10^5 - 4.8 \times 10^5$.

Other parameters include the threshold used to determine when to flood a measurement, and the minimum times between consecutive floodings of the measurement of the same link. These parameters are adjusted to the different VC turnover rates and the algorithms used.

7.4 The Test Network

The test network Net8 is depicted below.

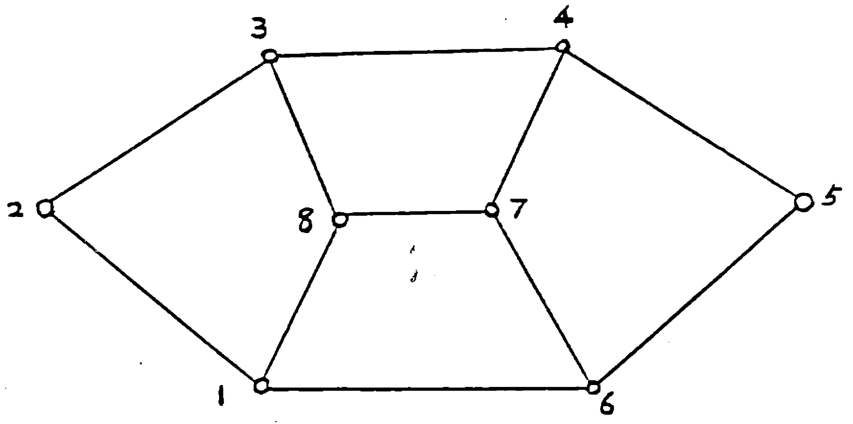


Figure 7.2 The Graph of Net8

The network has 16 OD pairs, each with a long-term average traffic rate of 300 bits or 30 packets per second. The average number of packets per VC per second is 0.1. The OD pairs and the capacities are given below.

OD pairs: $W = \{(1,3),(2,4),(3,5),(4,6),(5,7),(6,8),(7,1),(8,2),$

$(1,7),(2,8),(3,1),(4,2),(5,3),(6,4),(7,5),(8,6)\}.$

Capacities $C_{i,j} = 0.5$ kbits/sec, if i or j is in the set $\{7,8\}$,

otherwise $C_{i,j} = 1.0$ kbits/sec.

The above traffic pattern seems *balanced* and is used as our major simulation example. The concept of *balanced* traffic was perhaps first introduced by Chou et. al. [1981]. In their paper, a balanced traffic is a situation where each node sends equal amount of data traffic to every other node. According to this definition, the above traffic requirement is not *balanced*. However, there is no compelling reason to accept their definition literally. We shall call the above traffic requirement *balanced*.

7.5 Results and Interpretations

The simulation results confirm qualitatively with the analytical results developed earlier. However, we are more interested in knowing the properties of the routings that the analytical results cannot provide. In general, when the VC turnover rate is large, shortest path routing exhibits more oscillatory behaviors than gradient projection routing does. When the VC turnover rate is small, the two routings behave approximately the same. We also observe that the best gradient projection routing algorithm seems to be the algorithm 1 implemented with the deficiency metering rule.

In order to obtain realistic initial conditions, each run of simulation starts with an empty network – no packets and VCs. As the simulation progresses the VC arrives and leaves, the packets arrives and leaves. In this way we do not create artificial initial conditions for the routing algorithms. In section 7.5.2 we fail a link after the average load is around the long-term average load level. This is the only case when we 'artificially' disturb the network.

It is difficult to measure the long-term deviation from optimality accurately. One has to ask when a routing can be considered as having reached a steady-state. There is no good answer for it. Thus for a very crude measure, we take the average of the network costs from the time the load reaches approximately the long-term average level to the end of the simulation. Typical length of such a period is 400 seconds. We use this average network cost as a crude measure of the long-term average cost.

For both gradient projection and shortest path routing we try to 'optimize' their performance over the following parameters: minimum time between two consecutive floodings of a link, initial threshold and the decrements for the threshold used to determine when to initiate a flooding. In general, we find that the best operating conditions for both types of algorithms are about the same. As expected from the theory, the shortest path routing algorithm typically needs more frequent floodings than the gradient projection routing does. One interesting lesson learned from this kind of ad hoc optimization is that too many floodings do deteriorate the routing performance.

For the gradient projection algorithms, there is one extra parameter to optimize over, i.e., the stepsize α. As expected from nonlinear programming, too large a stepsize creates oscillation in the routing. Too small a stepsize, in a noisy routing environment, can force the routing to stagnate.

7.5.1 Balanced Traffic

For this part of the simulation, we use the traffic data provided in section 7.4. The traffic requirement is a situation which we considered as *balanced.*

We run ten experiments here, five with fast VC turnover rate and five with slow VC turnover rate. For each VC turnover rate, the five algorithms simulated are: the shortest path algorithm(S), algorithm 1 with the deficiency rule (M), algorithm 1 with the randomization rule (R), algorithm 1 with the fraction rule (MF), algorithm 2 with the deficiency rule (MG2). Recall that both algorithms 1 and 2 are the gradient projection algorithms with the orthant constraints. For fast VC turnover, the graphs presented in the appendix A start plotting the cost at time 300-th second and end at time 700-th second. For slow VC turnover, the graphs presented in the appendix A start plotting the cost at time 600-th second and end at time 1000-th second. Also note that the time unit used in these graphs is one tenth of a second, rather than a second. In general, the starting times are the first time the load reaches approximately the long-term average level. To give an approximate estimate of the long term deviation from optimality, we also provide the final costs, at the end time of each simulation.

Fast VC Turnover

The costs are listed below. The known optimal cost is about 42.0.

	S	M	R	MF	MG2
average cost	58.48	50.15	44.92	52.82	253.91
final cost	49.53	47.85	42.39	50.53	64.17

Some comments on the transient behaviors of the algorithms are in order. The shortest path algorithm (S) incurs higher cost almost everywhere than the gradient projection algorithm (M). More importantly, the shortest path algorithm produces wild oscillations for two occasions. This agrees with the intuition that the shortest path algorithm tends to over-react to good news (shortest paths) and bad news (long paths). The peaking in this shortest path routing simulation and other simulations are the result of one or more links reaching an utilization factor of 99% or above. Such kind of loading on a link is possible–since the optimal routing has a maximum utilization factor of 82%, if the load is temporarily high, the routing is not optimal, an utilization factor of 99% can be reached.

The difference between the deficiency rule and the fraction rule does not appear to be large. However, generally in this experiment and in other experiments, the fraction rule appears to be worse than the deficiency rule. The behavior of the gradient projection algorithm with the randomization rule in this simulation and in other experiments is quite puzzling. Except in the case of balanced traffic and slow VC turnover, the algorithm with the randomization rule incurs a lower cost most of the time than the algorithm with the deficiency rule. However, in all cases, the algorithm with the randomization rule produces peakings which the algorithm with the deficiency rule does not generate. Such kind of strange behaviors is not fully understood yet. Note that the randomization rule is an *open loop* control, i.e., it assigns VC independently of the current state of the paths. Such kind of non-adaptive open-loop policy may hurt the routing algorithm. The following is a scenario in which the randomization rule may hurt the performance. Suppose that, on the average, a number of VCs on a certain path p should depart from the network at the next routing update. However, by chance, these VCs do not depart, and the randomization rule, by chance again, assigns a group of new VCs on the path p. The possible outcome is that path p will be over-loaded temporarily.

Let us also recall the example provided in chapter 6 which shows that the deficiency rule does not anticipate VC departures and produces a worse quadratic cost than that of a rule anticipating the VC departures. Such kind of phenomenon may consistently hurt the performance of the deficiency rule.

The above two observations do not fully explain the different behaviors of the gradient projection algorithms with the randomization rule and the deficiency rule. Further study is needed to investigate the reasons and the implications of such behaviors.

Lastly, a comparison between algorithm 2 (MG2) and algorithm 1 (M) shows the clear superiority of algorithm 1. Intuitively, algorithm 2 is much less adaptive than algorithm 1 (compare equations (1) and (2)). Indeed, algorithm 1 uses the current flows to compute the next desired flows while algorithm 2 uses the current desired flows to compute the next desired flows. In general, the current flows can be quite different from the current desired flows when the algorithm has not converged to optimality. Hence algorithm 2 can often lead the routing into the wrong direction. In fact, algorithm 2 performs even worse than the shortest path.

Slow VC Turnover

The costs for this case are listed below.

	S	M	R	MF	MG2
average cost	49.79	49.95	58.98	51.26	59.12
final cost	52.82	52.51	58.11	54.34	55.18

With the VC now moves twice as slow as the fast case, shortest path algorithm should perform better, since now the update intervals become relatively twice shorter. In fact, the plots in the appendix reveal that there is no essential difference between shortest path routing and gradient projection routing. The shortest path algorithm incurs a higher cost in many places than that of the gradient projection algorithm. However, the two perform about the same, the shortest path algorithm ends up with a slightly lower average cost, while the gradient projection algorithm ends up with a lower final cost.

As for the gradient projection algorithm with the fraction rule (MF), it behaves just like its counterpart in the fast case. It still performs almost everywhere worse, but only by a small difference, than the algorithm with the deficiency rule (M).

With the slower VC turnover rate, algorithm 2 (MG2) seems better than its counterpart in the fast case. The oscillation is reduced. This may be due to the fact that, as the change in VC becomes slower, the algorithm, which is only adapted to the first and the second derivative lengths and the total load for each OD pair, adapts better.

Before discussing the simulations with *unbalanced* traffic we shall comment on the difference between the gradient projection algorithm (M) and the shortest path algorithm (S). It appears that when the VC turnover rate is fast, the shortest path algorithm generates peakings in the cost, while the gradient projection algorithm does not. However, the above simulations have the maximum utilization factor of 85%. The natural reason for the peakings generated by the shortest path algorithm is that, in one routing interval, all the new VCs are assigned to the shortest paths, and the shortest paths may then be overloaded temporarily. Thus if the above reason is correct, one should be able to see peakings generated by the shortest path algorithm at a lower load but with a higher VC turnover rate. Indeed, we run another set of simulations with VC turnover rate at about 30% and maximum utilization factor at about 75%. The costs are listed below.

	S	M
average cost	41.40	34.93
final cost	36.97	33.87

The graphs in the appendix show that the shortest path algorithm generates a peaking while the gradient projection algorithm performs better almost everywhere.

7.5.2 Unbalanced Traffic

In this section we change the traffic requirement in section 7.4 to create a more *unbalanced* situation. In the first set of experiments, we decrease the load on the left side (the nodes 1,2,3,8) of the network by 20% and increase the load on the right side (the nodes 4,5,6,7) of the network by 20%. In the second set of experiments, we reduce the capacity of link (7,6) and (6,7) to one half of their original values when the maximum utilization factor in network is about 85% loaded. This case is to simulate a link failure and create a distressed condition for the routing algorithm.

Unbalanced Left and Right Traffics

The VC arrival rates and departure rates are kept the same as that in the *balanced* traffic case. The only parameter changed is the average packet arrival rate per VC per second. The packet rates for OD pairs (1,3), (2,4), (2,8), and (8,2) are reduced to 80% of their original values; while the packet rates for OD pairs (4,6), (6,4), (7,5), and (5,7) are increased to 120% of their original values.

The average costs for the fast VC turnover rates are listed below. The known optimal cost is about 43.0.

	S	M	R	MF	MG2
average cost	52.68	49.62	46.97	51.66	144.82
final cost	54.30	53.64	46.01	56.57	60.19

There is again an evident difference between the shortest path algorithm (S) and the gradient projection algorithm(M). However, in this case the shortest path algorithm does not have wild oscillations. The shortest path algorithm still incurs a cost which is almost everywhere higher than the cost incurred by the gradient projection algorithm. Both algorithms do not seems to be bothered by the *unbalanced* load distribution.

The gradient projection algorithm with the randomized rule seems to do slightly worse than its counterpart in the *balanced* case. However, the behaviors are about the same. Likewise is the behavior of the algorithm with the fraction rule. It still performs worse than the algorithm with the deficiency rule most of the time.

An interesting case is the algorithm 2 which performs noticeably worse than its counterpart in the *balanced* case. It could be the unbalanced load or the random sequence that produces such a difference.

The average costs for the slow VC turnover are listed below.

	S	M	R	MF	MG2
average cost	48.33	49.75	47.14	49.94	55.62
final cost	45.70	46.20	39.64	47.86	46.14

There are no major differences between the graphs in this case and the graphs in the slow and *balanced* case. The only noticeable differences come from algorithm 2 (MG2) and the gradient projection algorithm with the randomization rule (R). There are slightly more peakings with the algorithm 2 in the *unbalanced* case than that in the *balanced* case. The algorithm with the randomization rule also has much less peakings than its counterpart in the *balanced* case. Note that in this case the shortest path algorithm preforms slightly better than the gradient projection algorithm. The difference is not significant though.

Link Failure

For this set of experiments we choose to study only the shortest path algorithm (S) and the gradient projection (algorithm 1 with the deficiency rule) algorithm (M). We also simulate only the fast VC turnover case, as this is a more distressed case than the corresponding slow VC turnover case.

At time 320-th second, the capacities for links (6,7) and (7,6) are reduced from 0.5 kbits/sec to 0.25 kbits/sec. One possible practical situation resembles this case is as follows. There are two transmission lines or media connecting nodes 6 and 7, and accidentally one of the lines or media goes down.

At time 320-th second, node 6 and 7 detect the link failure and start flooding a special message to all the other nodes saying that links (6,7) and (7,6) lose half of their capacities. Upon receiving these messages, all the nodes which have paths using these two links start shifting half of the original VCs using these links to the current shortest paths. While these VCs are being re-routed, all the links and paths involved in these re-routing will have their measurements updated. For example, if the link (2,1) discovers at time t that 10 VCs are suddenly joined to the groups of VCs using itself, all the past instantaneous flow measurements $f_{ij}(t-1), \ldots, f_{ij}(t-100)$ and the current flow measurement $F_{ij}(t)$ are increased by 10 units of average data traffic per VC per second.

After this re-routing is done, some links are operating around a utilization factor of 90%, and the routing becomes very susceptible to further disturbance. The oscillations we see in the graphs in the appendix are due to the new disturbance. The optimal cost for this case is about 64.7. The costs are listed below.

	S	M
average cost	456.04	369.56
final cost	376.60	87.94

As we can see from the graph, the gradient projection algorithm struggles a while before it finally reduces the congestion to a reasonable level. However, the shortest path algorithm has wilder and more oscillatory swings. Note that at time 700-th second, the gradient projection algorithm appears to converg to a neighborhood of an optimal routing while the shortest algorithm is still oscillating.

7.5.3 Synchronous Routing

One problem with the asynchronous routings simulated so far is that the controllers usually carry out a routing update with out-of-date flow measurements. Sometimes this out-of-date information can mislead the routing controllers. One way to remedy this problem is to use synchronous routing, i.e., all the flow measurements are flooded at the same time, and all nodes update their routings at the same time.

We only run these experiments for the fast VC turnover case with the *balanced* load. Below are the costs.

	Syn. S	Asyn. S	Syn. M	Asyn. M
average cost	86.98	58.48	48.04	50.15
final cost	53.93	49.53	45.81	47.85

We see that the shortest path asynchronous algorithm is better than the corresponding synchronous algorithm. This is perhaps due to the fact that the shortest path algorithm needs short update intervals, and our asynchronous shortest path algorithm has a shorter update interval than its synchronous counterpart. Here the availability of the asynchronously flooded flow measurement seems to help the shortest path algorithm reduce the oscillations. As for the gradient projection algorithms, the synchronous and asynchronous cases are about the same, with the synchronous algorithm incurring a slightly lower cost. This suggests that up-to-date flow information is good for the gradient projection algorithms.

CHAPTER EIGHT

EXTENSIONS

We discuss some possible extensions to the results of this thesis in this chapter. The following issues are discussed: shortest path routing with length functions taking values from a discrete set, the path assignment problem, and optimal routing in an integrated network.

We have seen that the shortest path algorithm may not converge to optimality when the length functions take values from a discrete set. However, the counter-example that we have shown assumes deterministic VC arrivals and departures. It is interesting to investigate whether or not a similar non-convergence can happen in the stochastic case. For data network applications, one simple way to remedy this non-convergence problem is to replace the discontinuous length functions by continuous ones. For nonlinear programming the counter-example in fact shows that a version of the Frank-Wolfe method fails to converge when the cost is non-differentiable. Thus it is interesting to find out what modification is needed to make the Frank-Wolfe method converge.

It is clear that the research on the path assignment problem, or in general, optimal dynamic routing problems, have a long way to go. It is interesting to find an optimal policy for the path assignment problem. However, this problem appears to be difficult. Thus it makes sense to consider ad hoc policies like the deficiency rule proposed in this thesis.

Two alternative policies to the deficiency rule are suggested below. First is that the arriving VC is assigned to the path which is, on the average, most deficient in the desired number of VCs at the next VC arrival. Second is that the arriving VC is assigned to the path which is, on the average, most deficient in the desired number of VCs at the next routing update. Since both policies involve estimating the VC arrival and departure rates, they are susceptible to the estimation errors. The second policy requires, in addition, the next routing update time, which may not be known in an asynchronously operating environment.

As the fraction rule is used in the industry, it is also interesting to find out the statistics of the path flows resulting from such a rule. We believe that the gradient projection algorithm implemented with the fraction rule should have a similar convergence result to that implemented with the deficiency rule. Possibly, the proof technique we used to find the bound for the quadratic cost incurred by the deficiency rule can be applied similarly to show this result. Such a proof technique may provide a useful tool to solve related queueing control problems.

Furthermore, our simulation results show that the gradient projection algorithm with the randomization rule performs in some way better than the algorithm with the metering rule in certain operating conditions. Further investigation is needed to understand this phenomenon and its implications.

Let us consider the overall optimal quasi-static routing problem. It is clear that the future communication networks will most likely be Integrated Services Networks (or simply integrated networks). In an integrated network, there can be many different kinds of user-pairs: interactive data exchange, voice conversation, video transmission, electronic mailing, and etc. Some of these user-pairs, e.g., digitized voice and video, demand small end-to-end delays. However, our formulation only accounts for the average delay per packet regardless of the type of user-pairs. The question one can ask is that is it possible to reformulate the problem to account for the end-to-end delay requirement for different kinds of user-pairs and solve the resulting optimal quasi-static routing problem. We believe that in this newly formulated problem the cost will not be a separable function of the total link flows, and as a result, the new problem is more difficult.

REFERENCES

Bertsekas, D.P.,(1976), "On the Goldstein-Levitin-Poljak Gradient Projection Methods," *Trans. on Auto. Control*, Vol AC-20, pp. 174-184.

Bertsekas, D.P.,(1979), "Algorithms for Nonlinear Multicommodity Network Flow Problems," *proc. int. Symposium on Systems Optimization and Analysis*, A. Bensoussan and J.L. Lion, (eds.), Springer-Verlag, N. Y., pp. 210-224.

Bertsekas, D.P.,(1980), "A Class of Optimal Routing Algorithms for Communication Networks," *proc. Fith Int. Conf. Comp. Comm.*, Atlanta, GA, Oct. pp. 71-75.

Bertsekas, D.P.,(1982a), "Projected Newton Methods for Optimization Problems with Simple Constraints," *SIAM J. Control and Optimization*, Vol 20, pp. 221-246.

Bertsekas, D.P.,(1982b), "Distributed Dynamic Programming," *Trans. on Auto. Control*, Vol AC-27, pp. 610-615.

Bertsekas, D.P.,(1982c), "Dynamic Behavior of Shortest Path Routing Algorithm for Communication Networks," *Trans. on Auto. Control*, Vol. AC-27, pp. 60-74.

Bertsekas, D.P.,(1982d), "Optimal Routing and Flow Control Methods for Communication Networks," in *Analysis and Optimization of Systems*, Bensoussan,A., and Lions, J.L., (eds.), Springer-Verlag, N.Y., pp.615-643.

Bertsekas, D.P.,(1983), "Distributed Asynchronous Computation of Fixed Points," *Mathematical Programming*, Vol. 27, pp. 107-120.

Bertsekas, D.P., and Gafni, E.M., (1983), "Projected Newton Methods and Optimization of Multicommodity Flows," *Trans. on Auto. Control*, Vol. AC-28, pp. 1090-1096.

Bertsekas, D.P., Gafni, E.M., and Gallager, R.G., (1984), "Second Derivative Alforithms for Minimum Delay Distributed Routing in Networks," *IEEE Trans. on Commun.*, Vol. COM-32, pp. 911-919.

Bertsekas, D.P., and Gallager, R.G.,(1986), *Data Networks*, Prentice Hall, Englewood Cliffs, N.J.

Bertsekas, D.P.,Gendron,B., and Tsai,W.K.,(1984), "Implementation of an Optimal Multicommodity Network Flow Algorithm Based on Gradient Projection and a Path Flow Formulation," *LIDS Report*, p-1364, MIT,MA.

Bertsekas, D.P., Tsitsiklis, J.N., and Athans,M.,(1984), "Convergence Theories of Distributed Iterative Processes: A Survey," *LIDS Report* P-1412, MIT, MA.

Boorstyn,R.R., and Livne,A.,(1981), "A Technique for Adaptive Routing in Networks," *IEEE Trans. on Commun.*, Vol. COM-29, pp.474-480.

Borkar, V.S., (1984), "On Minimum Cost Per Unit Time Control of Markov Chains," *SIAM J. Control and Optimization*, Vol. 22, Nov., pp. 965-978.

Cantor, D.G., and Gerla,M., (1974), "Optimal Routing in a Packet Switched Computer Network," *IEEE Trans. on Commun.*, Vol. C-23, pp. 1062-1069.

Chou, W., et. al., (1981), " The need for Adaptive Routing in the Chaotic and Unbalanced Traffic Environment," *IEEE Trans. on Commun.*, April 1981.

Cohen, A., Baraniuk, S., (1986), "OPNET User Manual," MIT, Dept. of EECS, Laboratory for Information and Decision Systems.

Dunn, J.C., (1979), "Rates of Convergence of Conditional Gradient Algorithms Near Singular and Nonsingular Extremals," *SIAM J. Control and Optimization*, Vol. 17, pp. 187-211.

Ephremides, A., (1978), "Extension of an Adaptive Distributed Routing Algorithms to Mixed Media Networks," *IEEE Trans. on Commun.*, Vol. COM-26, Aug, pp. 1262-1266.

Ephremides, A., Varaiya, P., and Walrand, J., (1980), "A Simple Dynamic Routing Problem," *Trans. on Auto. Control*, Vol. AC-25, Aug. , pp. 690-693.

Foschini., G.J., and Salz, J.,(1978), "A Basic Dynamic Routing Problem and Diffusion," *IEEE Trans. on Commun.*, Vol. COM-26, pp. 320-347.

Frank, H., and Chou, W.,(1971), "Routing in Computer Networks," *Networks* Vol. 1, pp. 99-122.

Fratta, L., Gerla, M., and Kleinrock, L.,(1973), "The Flow Deviation Method: An Approach to Store-and-Forward Communication Network Design," *Networks*, Vol. 3, pp. 97-133.

Gafni, E.M., and Bertsekas, D.P., (1983a), "Path Assignment for Virtual Circuit Routing," *SIGCOMM 83 Symposium on Communication Architectures and Protocols,* Austin, Texas, March, pp. 21-25.

Gafni, E.M., and Bertsekas,(1983b), "Asymptotic Optimality of Shortest Path Routing," *LIDS Report,* p-1307, MIT, MA.

Gallager, R.G.,(1977), "A Minimum Delay Routing Algorithm Using Distributed Computation," *IEEE Trans. on Commun.*, Vol. COM-25, pp. 73-85.

Goldstein, A.A., (1964), "Convex Programming in Hilbert Space," *Bull. Amer. Math. Soc.*, Vol. 70, pp. 709-710.

Hajek, B., (1982), "Optimal Control of Two Interacting Service Stations, "*Proc. 21st C.D.C.*, pp. 840-845.

Keilson, J. (1979), *Markov Chain Models–Rarity and Exponentiality*, Springer-Verlag , N.Y.

Kleinrock, L., (1964), *Communication Nets: Stochastic Message Flow and Delay*, McGraw-Hill, N.Y.

Kobayashi,H., and Konheim,A.G.,(1977), "Queueing Models for Computer Communication System Analysis," *IEEE Trans. on Commun.*, Vol. COM–25, pp.2-29.

Levitin, E.S., and Poljak, B.T., (1966), "Constrained Minimization Problems," *USSR Comput. Math. Math. Phys.*, Vol. 6, pp. 1-50.

Luenberger, D.G.,(1973), *Introduction To Linear and Nonlinear Programming*, Addison-Wesley, Reading, MA.

McQuillan, J.M., Richer, I., and Rosen, E.C., (1979), "The New Routing Algorithm for the ARPANET," *IEEE Trans. on Commun.*, Vol. COM-28, pp. 771-719.

McQuillan, J.M., and Walden, D.C., (1977), "The Arpa Network Design Decisions," *Computer Networks* 1, pp. 243-289.

Rishel, R., (1975), "A Minimum Principle for Controlled Jump Processes," *Control Theory Numerical Methods and Computer System Modeling*, Bensoussan A., and Lions J.L., (eds.), Springer-Verlag, Berlin, pp. 493-508.

Rockafellar,R.T.,(1970), *Convex Analysis*, Princeton Un. Press, Princeton, N.J.

Rosberg, Z., Varaiya, P.P., and Walrand, J.C., (1982), "Optimal Control of Service in Tandem Queues," *Trans. on Auto. Control*, Vol. AC-27, pp. 600-610.

Rudin, H., (1976), " On Routing and Delta Routing: A Taxonomy and Performance Comparison of Techniques for Packet-Switched Networks," *IEEE Trans. on Commun.*, Jan.

Sarachik, P.E., (1982), "An Effective Local Dynamic Strategy to Clear Congested Multi-Destination Networks," *Trans. on Auto. Control*, Vol. AC-27, pp. 510-513.

Sarachik, P.E., (1984), "Congstion Reducing Dynamic Routing Strategies for Multidestination Traffic Networks," *proc. 23rd C.D.C.*, Las Vegas, N.V., pp. 1383-1387.

Schwartz, M., and Cheung, C.K., (1976), "The Gradient Projection Algorithm for Multiple Routing in Message-Switched Networks," *IEEE Trans. on Commun.*. Vol . COM-24., pp. 449-456.

Schwartz, M., and Stern, T.E., (1980), " Routing Techniques Used in Computer Communication Networkes," *IEEE Trans. on Commun.*, Vol . COM-28, pp. 529-552.

Segall, A., (1977), "The Modeling of Adaptive Routing in Data-Communication Networks," *IEEE Trans. on Commun.*, Vol. COM-25, pp.85-95.

Segall, A., (1979), "Optimal Routing for Virtual Line-Switched Data Networks," *IEEE Trans. on Commun.*, Vol. COM-27.

Sobel, M.J., (1974), "Optimal Operation of Queues" in *Mathematical Method in Queueing Theory*, Clarke, A.B ., (ed,), Springer-Verlag, Berlin, pp. 231-261.

Stidham, S. and Prabhu, N.U., (1974), "Optimal Control of Queueing Systems," in *Mathematical Method in Queueing Theory*, Clarke, A.B., (ed.), Springer-Verlag, Berlin, pp. 231-361.

Stone, L., (1973), "Necessary and Sufficient Conditions for Optimal Control of Semi-Markov Jump Process," *SIAM J. Control and Optimization*, Vol. 11, pp. 187-201.

Tsitsiklis, J.N., (1984), "Problems in Decentralized Decision Making and Computation, " Ph.D. Theses, Dept. of Electrical Engineering and Computer Science, MIT, MA.

Tsitsiklis, J.N., and Bertsekas, D.P., (1985), "Distributed Asynchronous Optimal Routing in Data Networks," Manuscript.

Vastola, K. S., (1979), "A Numerical Study of Two Measures of Delay for Network Routing," M.S. Thesis, Dept. of Electrical Engineering, Univ. of Illinois, Urbana.

Yum, T.P., (1981), "The Design and Analysis of a Semi-dynamic Deterministic Routing Rule," *IEEE Trans. on Commun.*, Vol. COM-29, pp. 498-504.

Yum, T.P., and Schwartz, M., (1981), "The Join-Biased-Queue Rule and Its Application to Routing in Computer Communication Networks," *IEEE Trans. on Commun.*, Vol. COM-29, pp. 505-511.

APPENDIX A

GRAPHS OF SIMULATION RUNS

We present the graphs of the simulation runs described in section 7.5. The graphs appear in the order they appear in section 7.5. Recall the following abbreviations:

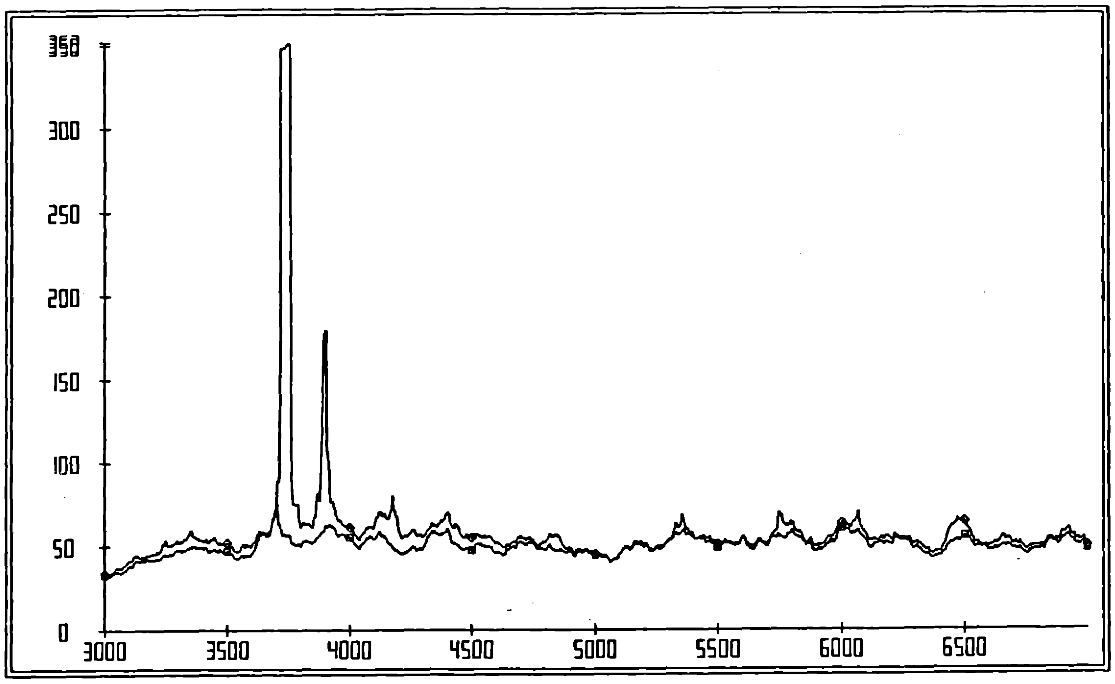
M – Gradient Projection Algorithm 1 with the Deficiency Rule.

R – Gradient Projection Algorithm 1 with the Randomization Rule.

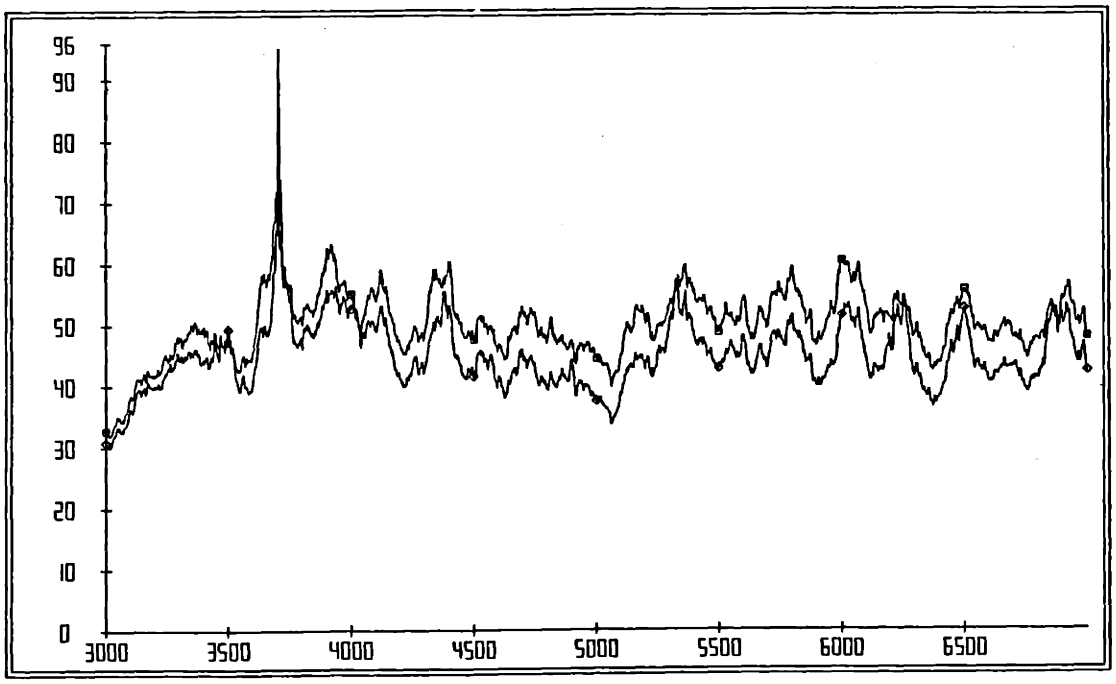
MG2 – Gradient Projection Algorithm 2 with the Deficiency Rule.

MF – Gradient Projection Algorithm 1 with the Deficiency Rule.

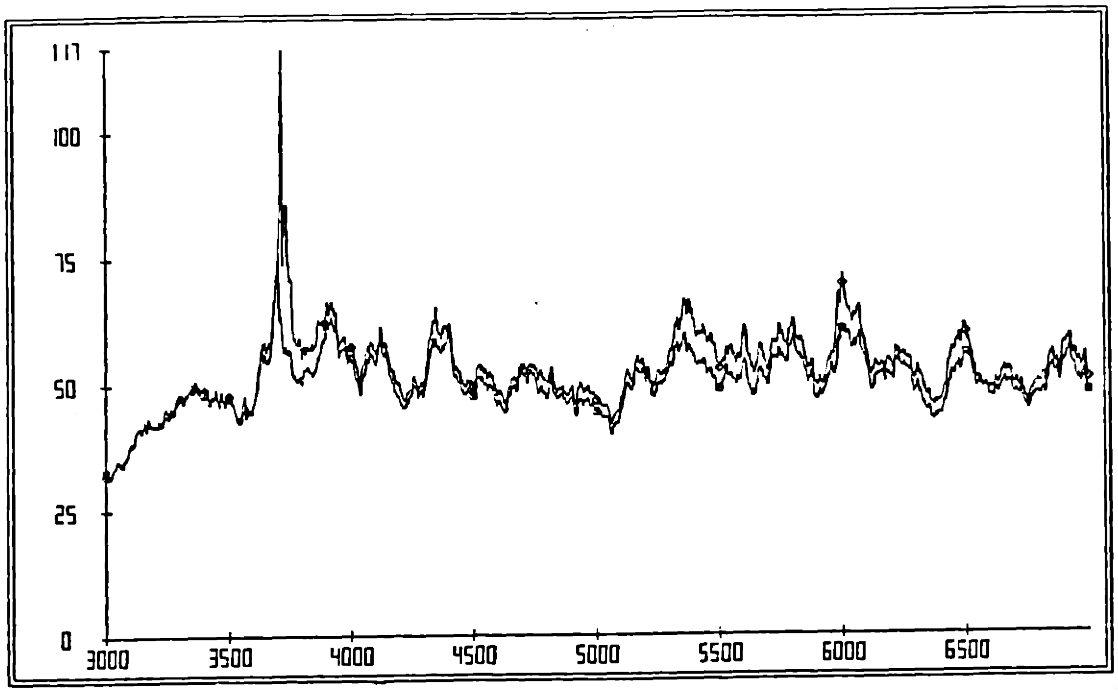
S – The Shortest Path Algorithm.



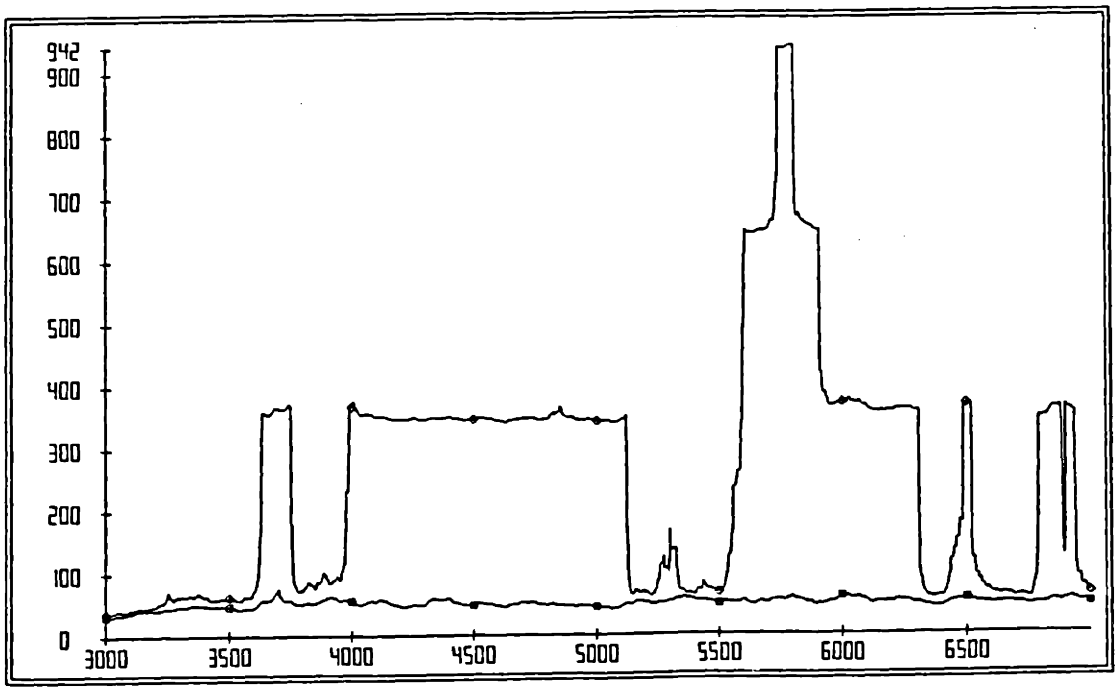
▫ Cost of Algorithm M with Fast VC Turnover and Balanced Traffic
◇ Cost of Algorithm S with Fast VC Turnover and Balanced Traffic



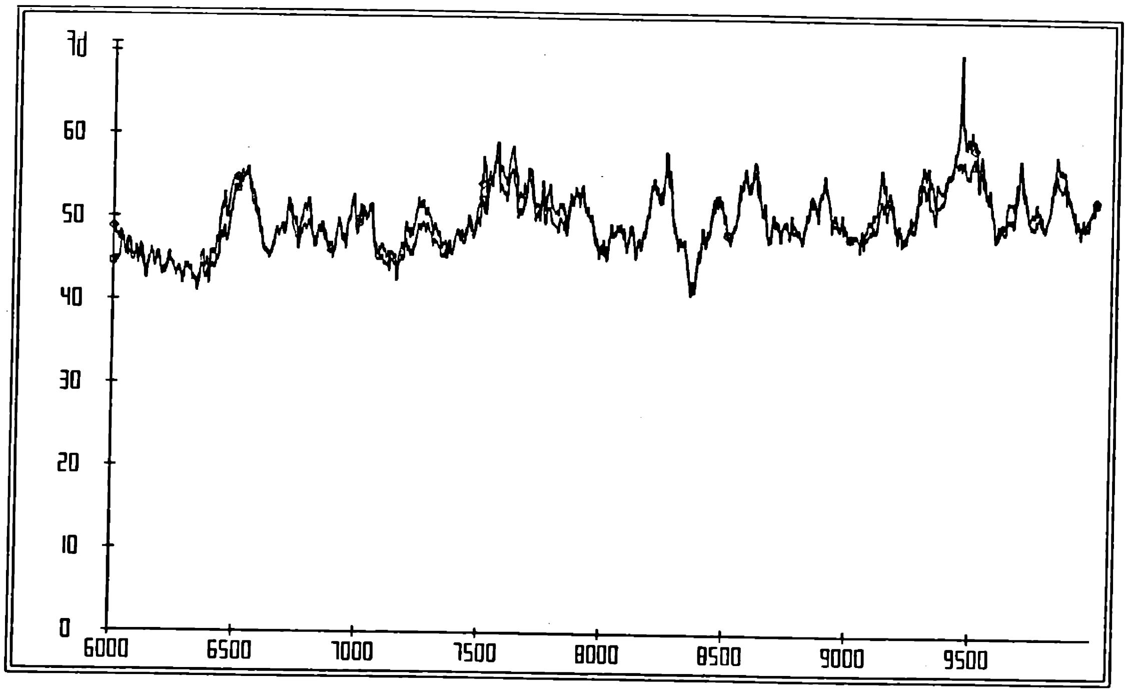
▫ Cost of Algorithm M with Fast VC Turnover and Balanced Traffic
◇ Cost of Algorithm R with Fast VC Turnover and Balanced Traffic



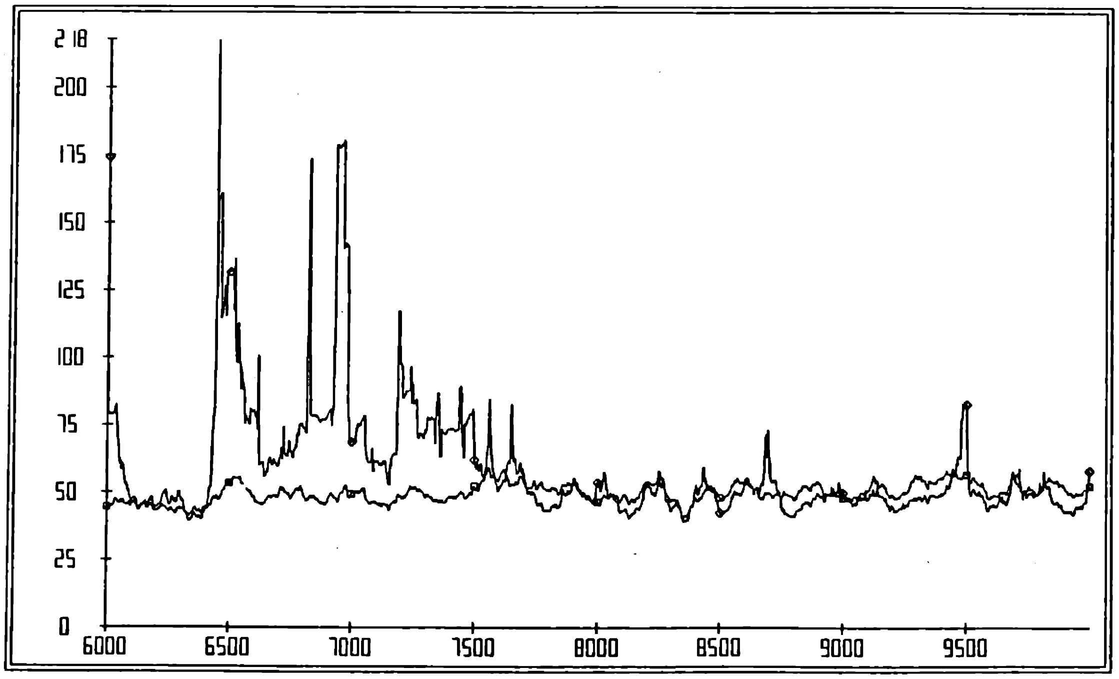
▫ Cost of Algorithm M with Fast VC Turnover and Balanced Traffic
⋄ Cost of Algorithm MF with Fast VC Turnover and Balanced Traffic



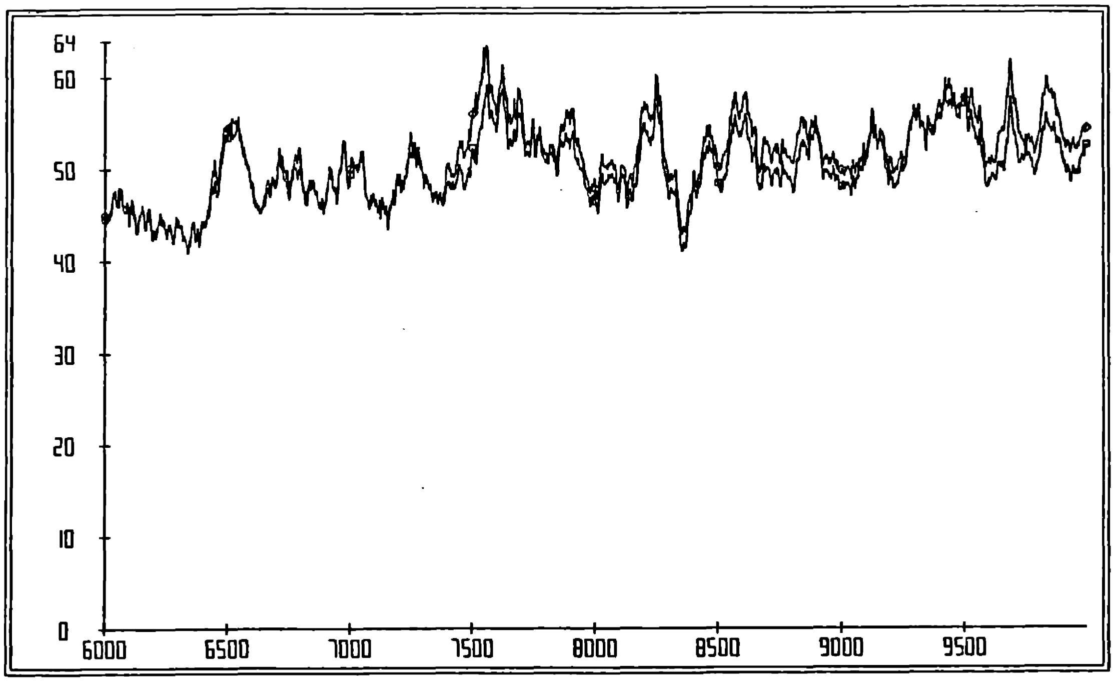
▫ Cost of Algorithm M with Fast VC Turnover and Balanced Traffic
⋄ Cost of Algorithm MC2 with Fast VC Turnover and Balanced Traffic



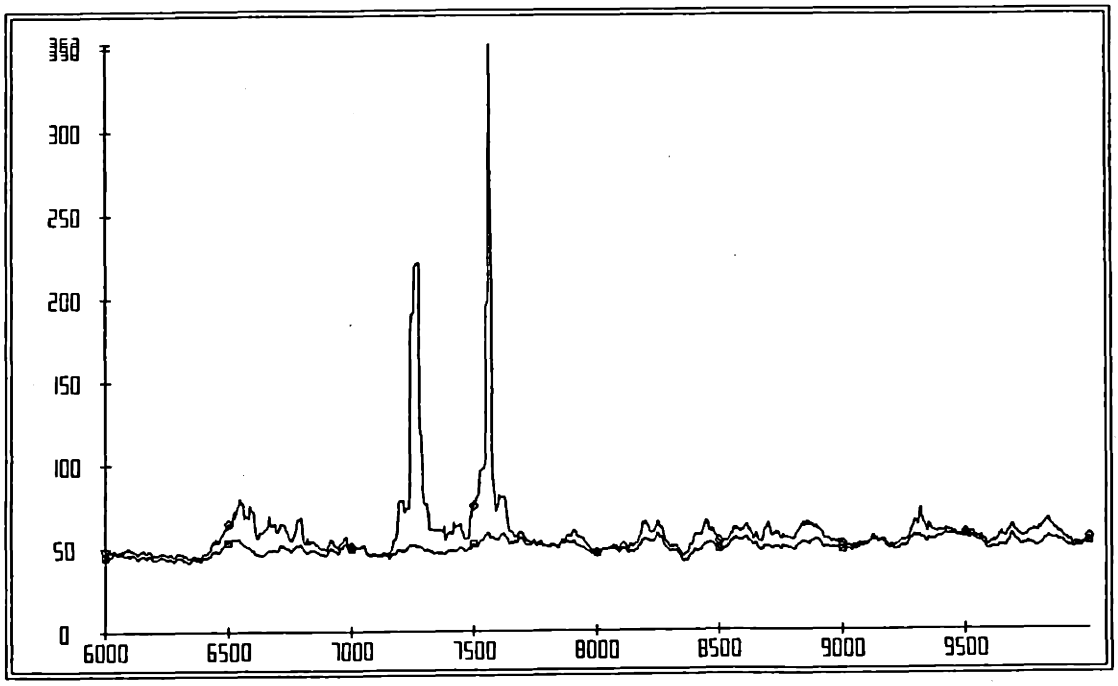
▫ Cost of Algorithm M with Slow VC Turnover and Balanced Traffic
◦ Cost of Algorithm S with Slow VC Turnover and Balanced Traffic



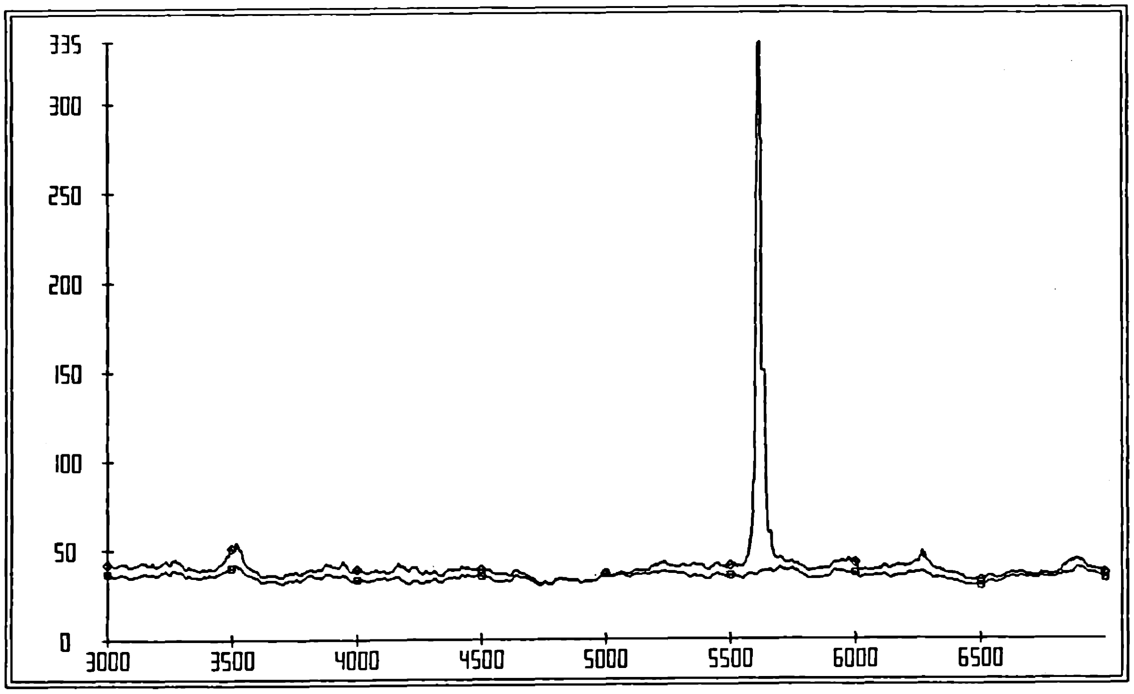
▫ Cost of Algorithm M with Slow VC Turnover and Balanced Traffic
◦ Cost of Algorithm R with Slow VC Turnover and Balanced Traffic



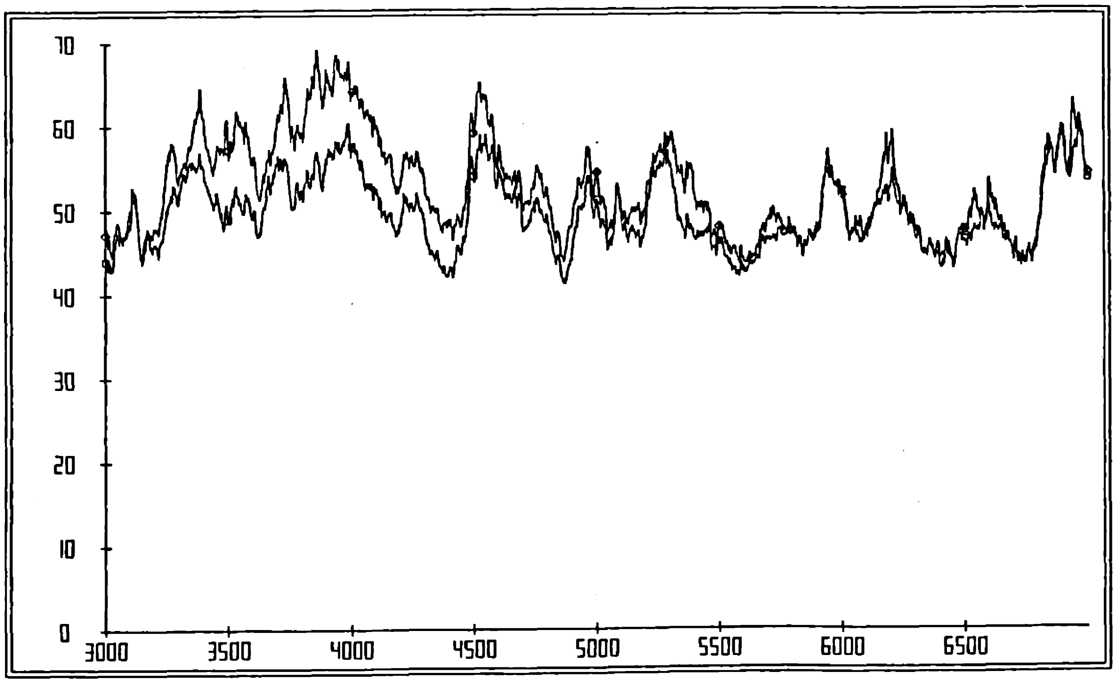
▫ Cost of Algorithm M with Slow VC Turnover and Balanced Traffic
◇ Cost of Algorithm MF with Slow VC Turnover and Balanced Traffic



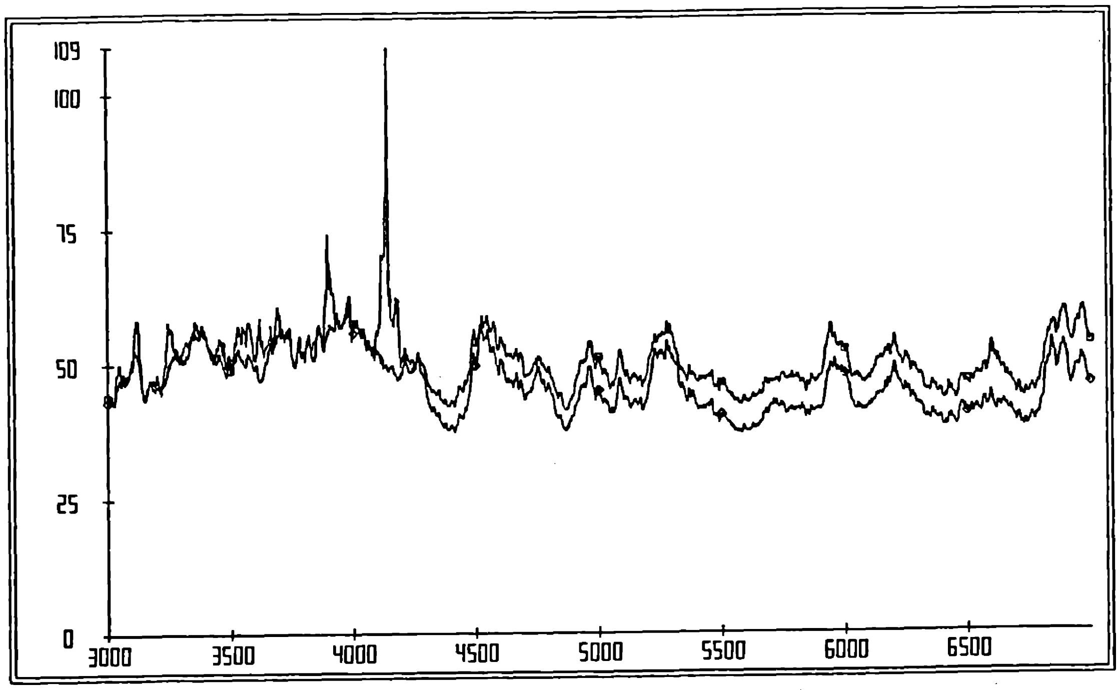
▫ Cost of Algorithm M with Slow VC Turnover and Balanced Traffic
◇ Cost of Algorithm MG2 with Slow VC Turnover and Balanced Traffic



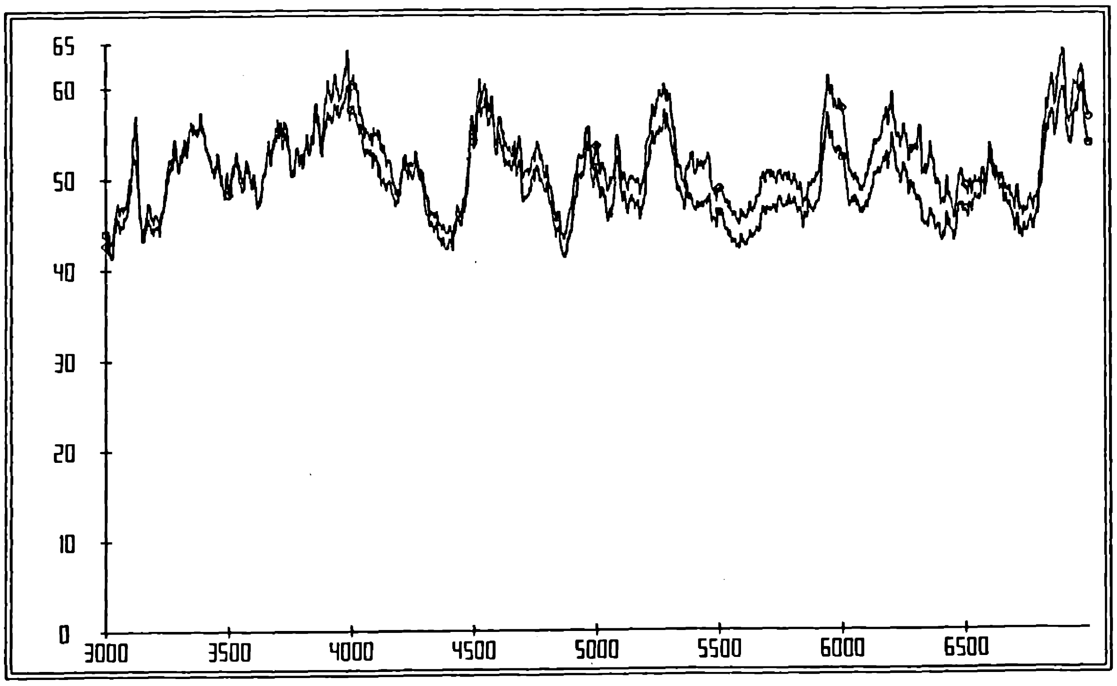
▫ Cost of Algorithm M with 30% VC Turnover and 75% Maxium Utilization
◇ Cost of Algorithm S with 30% VC Turnover and 75% Maxium Utilization



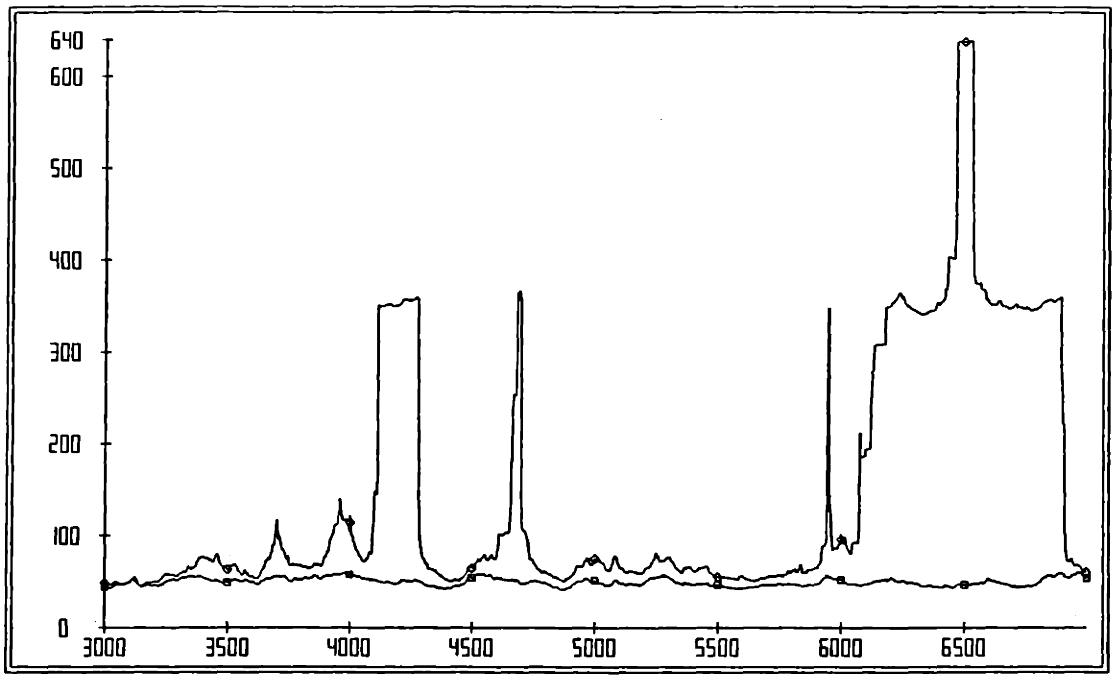
▫ Cost of Algorithm M with Fast VC Turnover and Unbalanced Traffic
◇ Cost of Algorithm S with Fast VC Turnover and Unbalanced Traffic



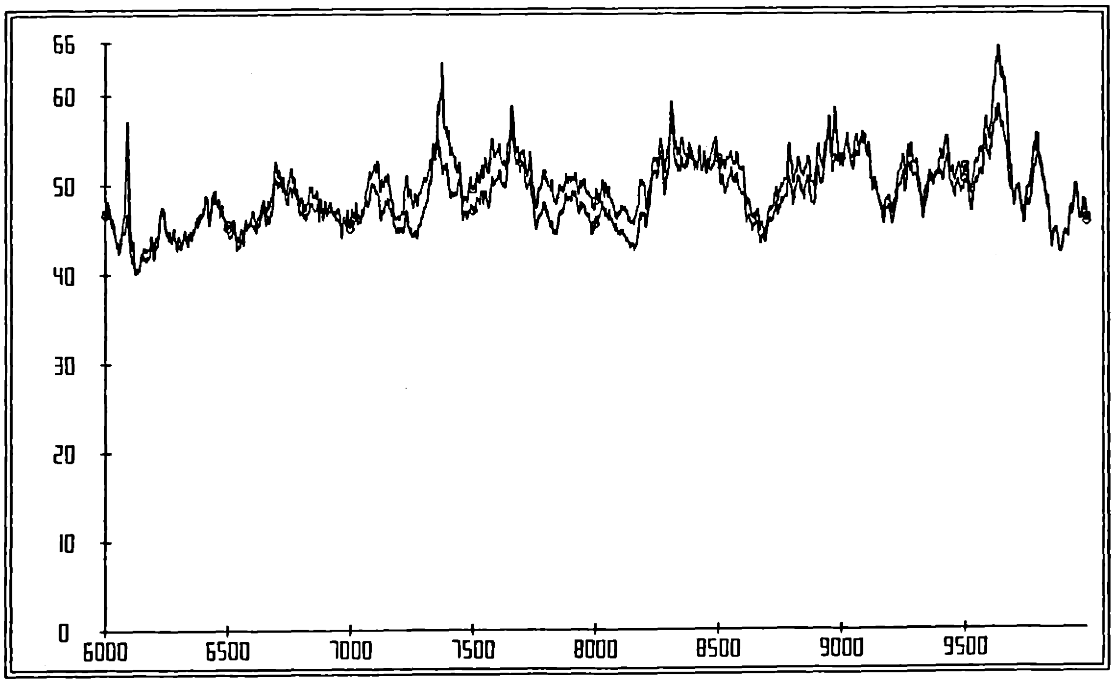
▫ Cost of Algorithm M with Fast VC Turnover and Unbalanced Traffic
◇ Cost of Algorithm R with Fast VC Turnover and Unbalanced Traffic



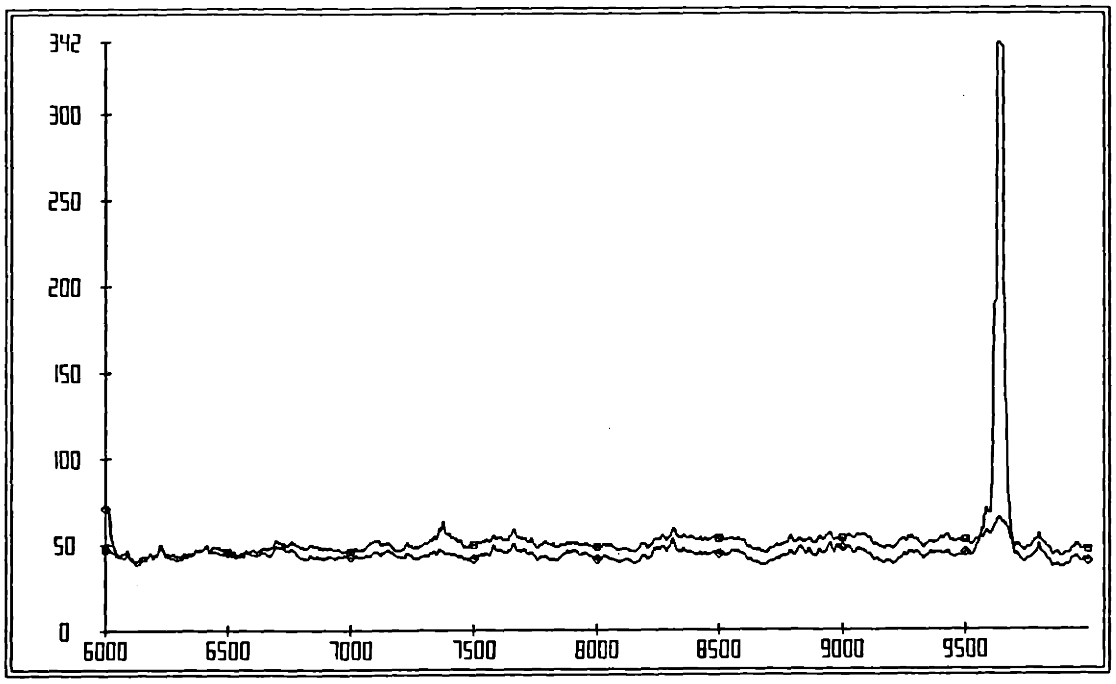
▫ Cost of Algorithm M with Fast VC Turnover and Unbalanced Traffic
◇ Cost of Algorithm MF with Fast VC Turnover and Unbalanced Traffic



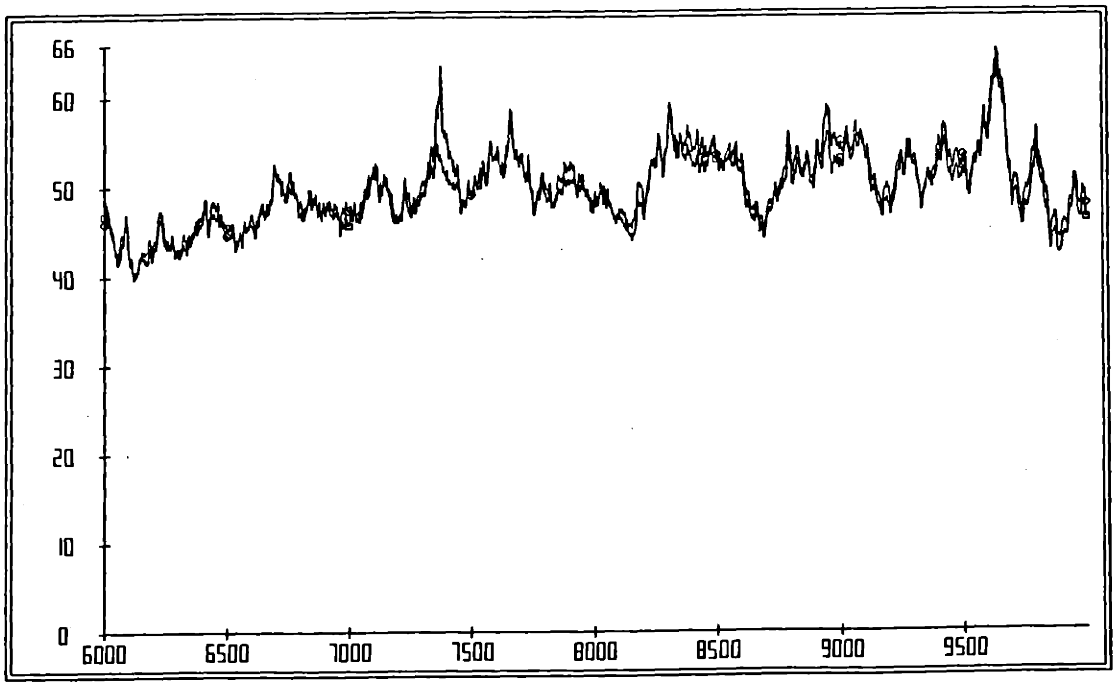
▫ Cost of Algorithm M with Fast VC Turnover and Unbalanced Traffic
◇ Cost of Algorithm MG2 with Fast VC Turnover and Unbalanced Traffic



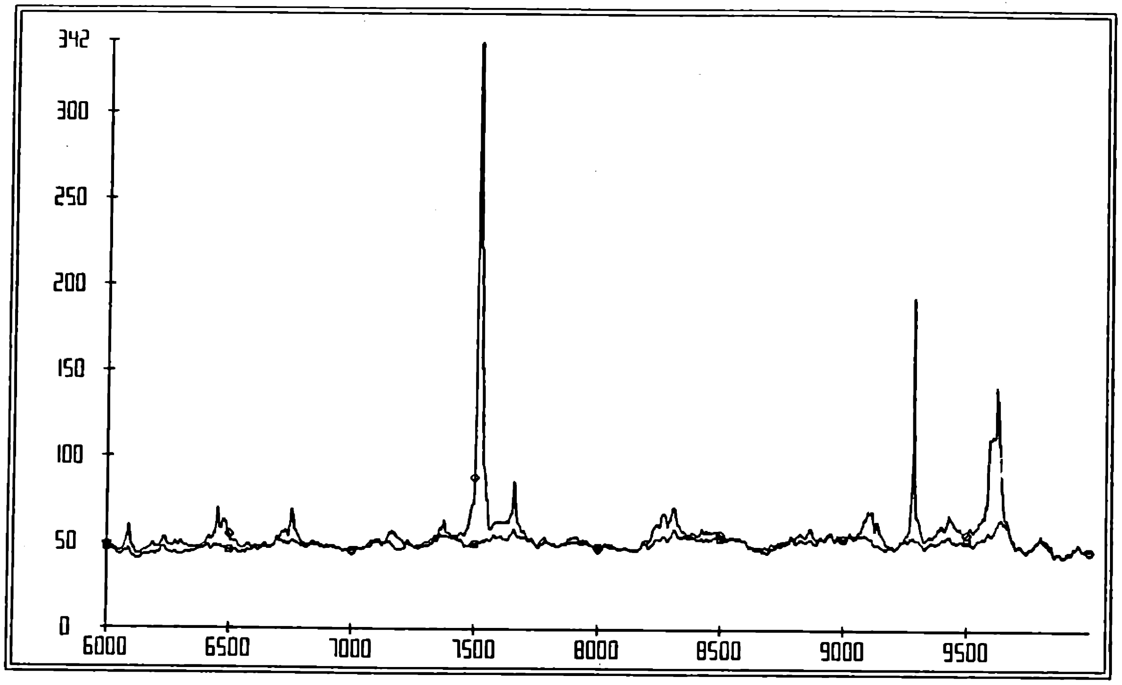
▫ Cost of Algorithm M with Slow VC Turnover and Unbalanced Traffic
◇ Cost of Algorithm S with Slow VC Turnover and Unbalanced Traffic



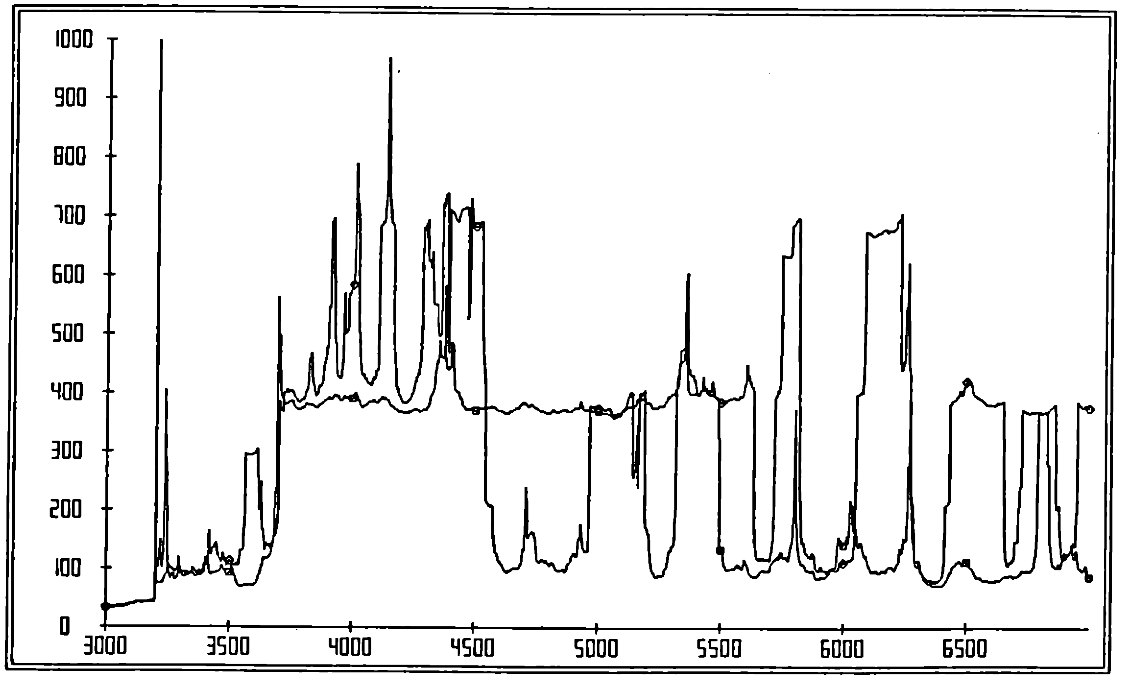
▫ Cost of Algorithm M with Slow VC Turnover and Unbalanced Traffic
◇ Cost of Algorithm R with Slow VC Turnover and Unbalanced Traffic



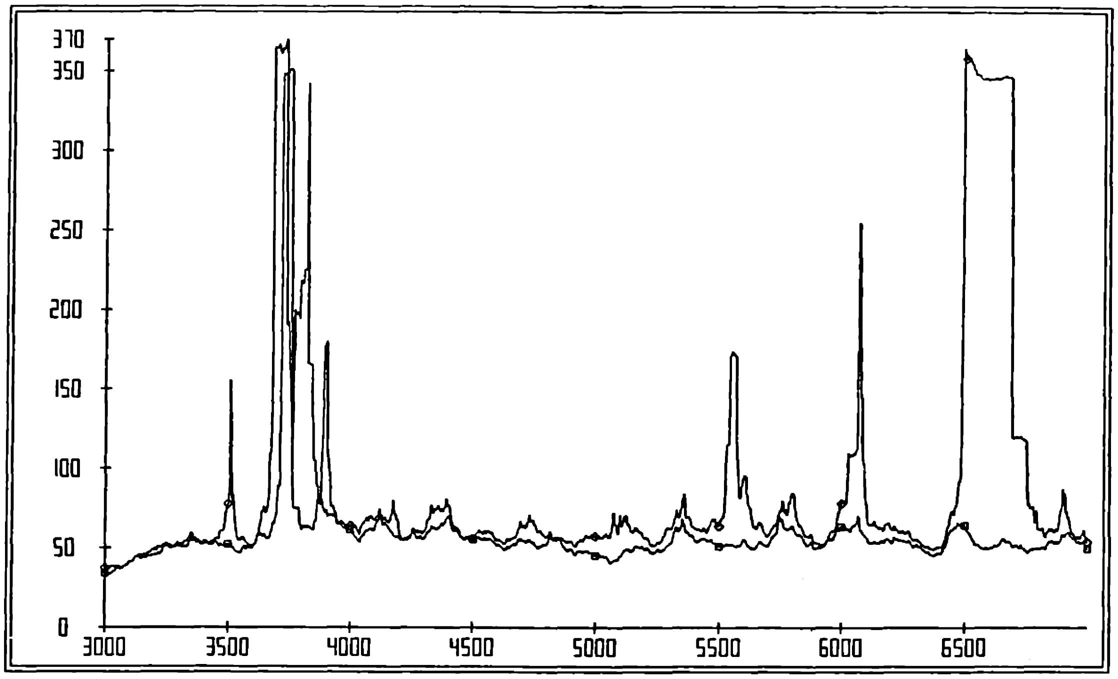
▫ Cost of Algorithm M with Slow VC Turnover and Unbalanced Traffic
◇ Cost of Algorithm MF with Slow VC Turnover and Unbalanced Traffic



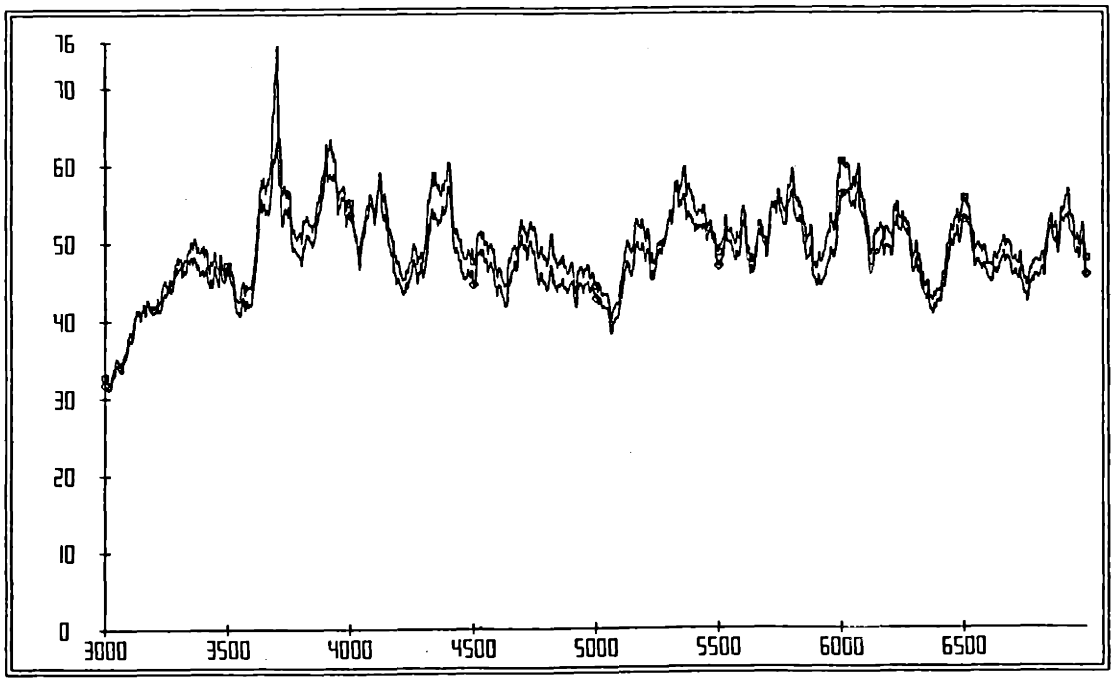
▫ Cost of Algorithm M with Slow VC Turnover and Unbalanced Traffic
◇ Cost of Algorithm MG2 with Slow VC Turnover and Unbalanced Traffic



▫ Cost of Algorithm M with Fast VC Turnover and Link Failure
◇ Cost of Algorithm S with Fast VC Turnover and Link Failure



▫ Cost of Asyhchronous Algorithm S with Fast VC Turnover and Balanced Traffic
◦ Cost of Syhchronous Algorithm S with Fast VC Turnover and Balanced Traffic



▫ Cost of Asyhchronous Algorithm M with Fast VC Turnover and Balanced Traffic
◦ Cost of Syhchronous Algorithm M with Fast VC Turnover and Balanced Traffic

APPENDIX B
SIMULATION CODES

We present a listing of the subroutines implementing the controllers, the routing processors, and the transmitters. Their functions are described in section 7.2. A transmitter is implemented by a cascade of a link server and a simple transmitter. The link server makes sure that the bit rate transmitted per simulation cycle does not exceed the pre-specified limit, and the simple transmitter just forwards the packets sent by the link server.

Since the codes are written as an extension of the OPNET simulator, these codes have to be used with the main body of the OPNET codes. The files cnet8.c and pnet8.c are given as an example showing how to use these codes. The network used in this example is Net8, our test network.

```
/*                          File struct.h
This file contains the data structures*/

#define maxp 10                         /*maximum no.  of paths*/
#define maxp1 11                        /*maximum no.  of paths plus 1*/
#define maxl 10                         /*maximum no.  of links in a paths*/
#define maxltot 30                      /*maximum no.  of links in a net*/
#define maxn 5                          /*maximum no.  of char.  in a name*/
#define maxint 100                      /*maximum length of flow update int.*/
#define maxRT 60                        /*maximum number of paths in a RT*/

typedef struct                          /* path */
        {
        int no_link;                    /*no.  of links*/
        int link_no[maxl];              /*link numbers of the path*/
        } path;

typedef struct                          /* ntcp */
        {
        double link_f[maxltot];         /*copy of the link flows*/
        int time[maxltot];              /*time stamps of the flows*/
        } ntcp;

typedef struct                          /* RT */
        {
        int no_receiver;                /*no.  of receivers*/
        int node_no;                    /*the node number*/
        int no_path;                    /*no.  of paths passing through*/
        int path_no[maxRT];             /*the paths which pass through*/
        int outch[maxRT];               /*the output channels :maximum
                                        no.  of paths through the node is 60*/
        } RT;

typedef struct                          /* OD */
        {
        char name[maxn];                /*ODname*/
        double p1;                      /*VC arrival prob.*/
        double p2;                      /*VC departure prob.*/
        double gm;                      /*gamma:  no.  of dbs per VC*/
        int no_path;                    /*no.  of paths*/
        double x[maxp];                 /*flow*/
        double xd[maxp];                /*desired flows*/
        double kd[maxp];                /*desired no.  of VC*/
        double k[maxp];                 /*no.  of VC*/
        double xt[maxp][maxint];        /*temp.  array of path flows*/
        double xbuf[maxp];              /*buffer to calculate path flows*/
        int i_first[maxp];              /*first index of the xt array*/
        int set_path_inch;              /*in channel of the paths*/
        int load;                       /*domain index for the load generator*/
        int set_path_pno[maxp];         /*path numbers*/
        int set_path_outch[maxp];       /*out channels of paths*/
        path path[maxp];                /*path list*/
        int i_sh;                       /*The shortest path no.*/
        int p_update_int;               /*path measurement interval for the OD*/
        int r_update_T;                 /*route update interval for the OD*/
        int r_update_bias;              /*route update bias for the OD*/
        int r_update_time;              /*next r_update time*/
        int update_flag;                /* =1 if the current time is a r_update
                                                time; =0 otherwise*/
        int dbsize;                     /*size of an ordinary db*/
        int old_update_t;               /*old routing update time*/
        } OD;

typedef struct                          /* node */
        {
        int no_link;                    /*no.  of links attached to the node*/
        int update;                     /* > 0 if arrival of control db*/
        int link_no[maxp][3];           /*link_no[.][0] = incoming link no;
                                          link_no[.][1] = outgoing link no.
                                          link_no[.][2] = outgoing channel no.*/
        int send_int;                   /*the interval for sending link flows*/
        int send_bias;                  /*the bias for sending link flows*/
        ntcp *ntpt;                     /*pt.  to a copy of the network*/
        RT *RTpt;                       /*pt.  to the Routing Table*/
        int no_OD;                      /*no.  of OD pairs*/
        OD *ODpt[2];                    /*pt.  to the OD pairs*/
        } node;

typedef struct                          /* Net */
        {
        int tm;                         /*minimum time between two floodings*/
        int tl;                         /*maximum time between r update*/
        int synch;                      /*=1 if synchronous routing*/
        int moving_a;                   /*=1 if moving avg; else exponential*/
        double beta;                    /*the weighting of the new measure.*/
        double m_thre;                  /*threshold for emergency flooding*/
        int linktot;                    /*total no.  of links*/
        int constant_db;                /*=1 if no.  of dbs per VC is constant*/
        int instant;                    /*=1 if all info.  is instantaneous*/
        int re_route;                   /*=1 if VC is rerouted ;=0 otherwise*/
        int det;                        /*=1 if no.  of VC's fixed;=0 otherwise*/
        int i_first[maxltot];           /*first index of the ft array*/
        char link_name[maxltot][maxn];  /*link name list*/
        double link_f[maxltot];         /*link flows*/
        double link_fo[maxltot];        /*old link flows*/
        int old_flood_t[maxltot];       /*old flood time*/
        double c[maxltot];              /*capacities*/
        int update_int;                 /*update interval for links*/
        double diff[maxltot];           /*reducing threshold for links*/
        double link_ft[maxltot][maxint];/*temp.  array of link flows*/
        int mode;                       /*mode for routing :  1=metering
                                          2=randomization,3=shortest path
                                          4= metering using fractions*/
        int algo;                       /*1= gradient proj wrt current flows
                                          2= gradient proj wrt desired flows*/
        char foutn[16];                 /*file name to output result*/
        int print_flag;                     /*flag=1:print all times;=0 no print*/
        int c_dbsize;                       /*size of a control db*/
        int print_init;                     /*=1 if print init data; =0 otherwise*/
        double f_thre;                      /*threshold for link congestion*/
        double small;                       /*decrement for threshold  for links*/
        double af;                          /*alfa the stepsize*/
        double cost;                        /*cost of the Network*/
        double load;                        /*load of the Network*/
        } Net;

typedef struct                              /* state_var */
        {
        Data_Block      held_db;            /*db being held*/
        int             active;             /*1 if active db held*/
        } state_var;
/*                      File con.c
This file contains the routing controllers' codes.
NCC (the Network Control Center):
        updates the network state,
        also serves as a local controller.
cn (the local controller at a node):
        does the follwings:
                controlling VC arrivals and departures,
                performing routing updates,
                and, initializing itself.*/

#include <stdio.h>
#include <opnet.h>
                        /* global declaration */
#include "struct.h"

extern  Net *Netpt;
extern FILE *fout;

                        /* controller NCC */

NCC(end,Ntpt,ndpt,fnet,fn,ODpt1,ODpt2,ntpt,RTpt)
        node* ndpt;
        Net *Ntpt;
        char *fn,*fnet;
        OD *ODpt1,*ODpt2;
        ntcp *ntpt;
        RT *RTpt;
        int end;
        {
        if(Time!=0)
                {
                Net_update();               /*update Net link flows*/
                if(Netpt->print_flag==1)    printNet();/*if needs to print
                                                    the network state*/
                Netpt->load = 0.0;
                nd_update(ndpt);            /*update the node*/

                if(Time==end)
                        fclose(fout);
                }
        else
                {
                ininet(Ntpt,fnet);              /*initialize the Network*/
                Netpt->load = 0.0;
                inind(ndpt,fn,ODpt1,ODpt2,ntpt,RTpt);/*initialize the node*/
                }
        }

                        /*Controller type cn*/

cn(ndpt,fn,ODpt1,ODpt2,ntpt,RTpt)
        node* ndpt;
        char* fn;
        OD *ODpt1,*ODpt2;
        ntcp *ntpt;
        RT *RTpt;
        {

        if(Time!=0)
                {
                nd_update(ndpt);                /*update the node*/
                }
        else
                {
                inind(ndpt,fn,ODpt1,ODpt2,ntpt,RTpt);/*initialize the node*/
                }

        }
/*                      File init.c
This file contains the initializing routines.
ininet:  initializes the network in general;
inind:  initializes a node;
iniOD:  initializes an OD pair.*/

#include <stdio.h>
#include <opnet.h>
#include "struct.h"
#define FPRINTF if(Netpt->print_init==1) printf(/*if print_init=1 print the data
                                                 as they are being read in */
extern FILE *fout;
extern Net *Netpt;

        /*This function initializes a network*/

ininet(Ntpt,fn)
        Net *Ntpt;
        char *fn;       /*The Network Data File*/
        {
        FILE *fp,*fopen();
        int i,imax,t,j;
        double f_thre;

        Netpt=Ntpt;     /*initializes the Network pointer Netpt*/

        fp=fopen(fn,"r");/*fp points the network data file*/

                /*start reading in network data*/

        fscanf(fp,"%d %d %d %d", &(Netpt->instant),
                &(Netpt->constant_db),&(Netpt->re_route),&(Netpt->det));
        fscanf(fp,"%s %d",Netpt->foutn,&(Netpt->print_flag));
        fscanf(fp,"%d %d",&(Netpt->c_dbsize),&(Netpt->print_init));
        fout=fopen(Netpt->foutn,"w");
        FPRINTF       "instant = %d\n",Netpt->instant);
        FPRINTF       "control dbsize = %d print_init = %d constant_db = %d\n",
                Netpt->c_dbsize,Netpt->print_init,Netpt->constant_db);
        FPRINTF       "re_route =  %d det = %d\n",Netpt->re_route,Netpt->det);
        fscanf(fp,"%d %d %f",&(Netpt->synch),&(Netpt->moving_a),&(Netpt->beta));
        fscanf(fp,"%d %d %f %f",&(Netpt->tm),&(Netpt->tl),
                &f_thre,&(Netpt->small));
        fscanf(fp,"%f",&(Netpt->m_thre));
        FPRINTF    "m_thre = %f\n",Netpt->m_thre);
        Netpt->f_thre = f_thre;
        FPRINTF    "tm =%d tl =%d f_thre =%f small =%f\n",
        Netpt->tm,Netpt->tl,Netpt->f_thre,
                Netpt->small);
        FPRINTF    "synch = %d moving_a = %d beta = %f\n",
                Netpt->synch,Netpt->moving_a,Netpt->beta);
        fscanf(fp,"%d %d",&(Netpt->mode),&(Netpt->algo));
        FPRINTF       "Output File :  %s Print-Flag = %d\n",
                Netpt->foutn,Netpt->print_flag);
        FPRINTF       "mode = %d algo = %d\n",Netpt->mode,Netpt->algo);
        fscanf(fp,"%f",&(Netpt->af));
        FPRINTF       "af=%f\n",Netpt->af);

        fscanf(fp,"%d %d",&imax,&(Netpt->update_int));
        FPRINTF       "linktot = %d update_int = %d\n",imax,Netpt->update_int);
        Netpt->linktot=imax;
        for(i=0;i<imax;++i)
                {
                fscanf(fp,"%s %f",Netpt->link_name[i],&(Netpt->c[i]));
                FPRINTF       "link_name[%d]=%s c[%d]=%f\n",
                        i,Netpt->link_name[i],i,Netpt->c[i]);
                Netpt->link_f[i]=0.0;
                Netpt->link_fo[i] = 0.0;
                Netpt->i_first[i] = 0;
                Netpt->old_flood_t[i] = 0;
                Netpt->diff[i] = f_thre;
                for(j=0;j<t;++j)
                        {
                        Netpt->link_ft[i][j]=0.0;
                        }
                }
        fclose(fp);
        }

        /* This function initializes an OD pair*/

iniOD(ODpt,fp)
        OD *ODpt;
        FILE *fp;       /*pointer to the file containing the OD pair data*/
        {
        int i,imax, j, jmax,t;
        double gm,x,xtot,x1,x2;

        ODpt->update_flag=0;

                /*start reading in the OD pair data*/

        fscanf(fp,"%s",ODpt->name);
        fscanf(fp,"%d",&(ODpt->p_update_int));
        fscanf(fp,"%d %d",&(ODpt->load),&(ODpt->dbsize));
        FPRINTF       "\n OD pair :  %s p_update_int=%d load_index = %d\n",
                        ODpt->name,ODpt->p_update_int,ODpt->load);
        FPRINTF       "db_size = %d\n",ODpt->dbsize);

        fscanf(fp,"%f %f %f",&(ODpt->p1),&(ODpt->p2),&gm);
        FPRINTF       "p1=%f p2=%f gm=%f\n",ODpt->p1,ODpt->p2,gm);
        ODpt->gm=gm;
        fscanf(fp,"%d",&(ODpt->i_sh));
        FPRINTF       "i_sh=%d\n",ODpt->i_sh);

        fscanf(fp,"%d %d",&(ODpt->r_update_T),&(ODpt->r_update_bias));
        FPRINTF       "r_update_T = %d r_update_bias = %d\n",
                ODpt->r_update_T,ODpt->r_update_bias);
        ODpt->r_update_time = ODpt->r_update_T + ODpt->r_update_bias;
        ODpt->old_update_t = 0;

        /*calculate initial flows*/

        fscanf(fp,"%d",&(ODpt->no_path));
        FPRINTF       "no_path=%d\n",ODpt->no_path);
        imax = ODpt->no_path;
        x1=gm*(ODpt->dbsize);
        xtot=0.0;
        t=ODpt->p_update_int;
        for(i=0;i<imax;++i)
                {
                fscanf(fp,"%f",&(ODpt->k[i]));
                FPRINTF       "k[%d]=%f\n",i,ODpt->k[i]);
                xtot += ODpt->k[i]*x1;
                for(j=0;j<t;++j)
                        {
                        ODpt->xt[i][j] = 0.0;
                        }
                }

        for(i=0;i<imax;++i)
                {
                fscanf(fp,"%f",&(ODpt->kd[i]));
                FPRINTF       "kd[%d]=%f\n",i,ODpt->kd[i]);
                }

        /*set initial conditions*/

        for(i=0;i<imax;++i)
                {
                ODpt->xd[i] = ODpt->kd[i]*x1;
                ODpt->x[i] = 0.0;
                ODpt->i_first[i] = 0;
                }

        /*input path link description*/

        for(i=0;i<imax;++i)
                {
                fscanf(fp,"%d",&jmax);
                FPRINTF       "no_link[%d]=%d\n",i,jmax);
                ODpt->path[i].no_link=jmax;
                for(j=0;j<jmax;++j)
                        {
                        fscanf(fp,"%d",&(ODpt->path[i].link_no[j]));
                        FPRINTF       "path[%d].link_no[%d]=%d\n",
                                i,j,ODpt->path[i].link_no[j]);
                        }
                }

        imax = ODpt->no_path;
        fscanf(fp,"%d",&(ODpt->set_path_inch));
        FPRINTF       "IN CHANNEL = %d\n",ODpt->set_path_inch);
        for(i=0;i<imax;++i)
                {
                fscanf(fp,"%d %d",&(ODpt->set_path_pno[i]),
                        &(ODpt->set_path_outch[i]));
                FPRINTF       "set_path_pno[%d]=%d set_path_outch[%d]=%d\n",
                        i,ODpt->set_path_pno[i],
                        i,ODpt->set_path_outch[i]);
                }

        }

        /*This routine initializes a node*/

inind(ndpt,fn,ODpt1,ODpt2,ntpt,RTpt)
        node *ndpt;
        ntcp *ntpt;
        RT *RTpt;
        char *fn;       /*this file contains data for the node*/
        OD *ODpt1;
OD *ODpt2;
{
int i,imax,j,t,id;
FILE *fp;
FILE *fopen();

fp=fopen(fn,"r");

        /*start reading in the node data*/

FPRINTF     "\n Node file :  %s\n\n",fn);
fscanf(fp,"%d",&(ndpt->no_OD));
FPRINTF     "no.  of OD pairs = %d\n",
        ndpt->no_OD);

/*initializes the OD pairs originating from the node*/

ndpt->update = 0;
if(ndpt->no_OD == 1)
        {
        iniOD(ODpt1,fp);
        ndpt->ODpt[0] = ODpt1;
        }
else if(ndpt->no_OD == 2)
        {
        iniOD(ODpt1,fp);
        ndpt->ODpt[0] = ODpt1;
        iniOD(ODpt2,fp);
        ndpt->ODpt[1] = ODpt2;
        }

fscanf(fp,"%d %d %d",&(ndpt->no_link),
        &(ndpt->send_int),&(ndpt->send_bias));
FPRINTF     "\n no.  of links = %d :  send-int = %d send-bias = %d\n",
        ndpt->no_link,ndpt->send_int,ndpt->send_bias);

ndpt->ntpt=ntpt;        /*initializes the node pointer*/

imax = ndpt->no_link;
for(i=0;i<imax;++i)
        {
        fscanf(fp,"%d %d %d",&(ndpt->link_no[i][0]),
                &(ndpt->link_no[i][1]),&(ndpt->link_no[i][2]));
        FPRINTF     "inlink[%d][0]=%d outlink[%d][1]=%d outch[%d]=%d\n"
               ,i,ndpt->link_no[i][0],i,ndpt->link_no[i][1],
                i,ndpt->link_no[i][2]);
        }

/*initialize the Routing Tables*/

ndpt->RTpt = RTpt;

fscanf(fp,"%d %d %d",&(RTpt->no_receiver),
                &(RTpt->no_path),&(RTpt->node_no));
        FPRINTF       "no_receiver = %d no_path = %d\n",
                RTpt->no_receiver,RTpt->no_path);
        FPRINTF       "node_no = %d\n",RTpt->node_no);

        imax = RTpt->no_path;
        for(i=0;i<imax;++i)
                {
                fscanf(fp,"%d %d",
                        &(RTpt->path_no[i]),&(RTpt->outch[i]));
                FPRINTF       "path_no[%d] = %d outch[%d] = %d\n",
                        i,RTpt->path_no[i],i,RTpt->outch[i]);
                }

        /* Dynrt the ODpairs */

        imax = ndpt->no_OD;
        for(i=0;i<imax;++i)
                {
                dynrt(ndpt,ndpt->ODpt[i]);
                }

        fclose(fp);
        }
```

```
/*                          File dynrt.c
This file contains the routine controlling the VCs--dynrt--and
the routines responsible for updating the states of the network,
the nodes, and the OD pairs.
dynrt (DYNamic RouTing):  generates and destroys VCs, and assigns paths;
re_route:  re-routes the VCs;
Net_update:  updates the the network link flows;
nd_update:  updates the states of a node;
OD_update:  updates the path flows for an OD pair.*/

#include <stdio.h>
#include <opnet.h>
#include <math.h>
#include "struct.h"

#define SET set_generator_load(ODpt->load,constant((int) xflow)); else
#define NONZERO if(pp != 0.0)

extern FILE *fout;
extern Net *Netpt;

        /*This function assigns VC dynamically*/

dynrt(ndpt,ODpt)
        OD *ODpt;
        node *ndpt;
        {
        double pp,rand1(),ran1,xflow,ktot,dif,mindif;
        double p[maxp1],p1[maxp1];
        double ran;
        int i,imax,imax1,i_d,mode;

        /*if need to re-route the VC's*/

        if(Netpt->re_route)
                {
                re_route(ODpt);
                goto prin;
                }

        /*calculate prob.  so that ran can be tested*/

        mode = Netpt->mode;
        ran=rand1();
        ran1=rand1();
        imax=ODpt->no_path;
        pp=ODpt->p2;
        p[0]=ODpt->k[0]*pp;
        ktot = ODpt->k[0];

        for(i=1;i<imax;++i)
                {
                ktot += ODpt->k[i];
                p[i]=p[i-1]+ODpt->k[i]*pp;
                }

        p[imax]=p[imax-1]+ODpt->p1;
        if(p[imax]>1)
                printf("probability too large\n");

        /*test which case:  updating VC nos.*/
        /*i_d is the path to add the arriving VC*/

        if(ran<p[0])                        /*path[0] has a departure of VC*/
                {
                        if(ODpt->k[0]>=1.0)
                        {
                        ODpt->k[0]=ODpt->k[0]-1.0;
                        xflow = (ktot - 1.0)*(ODpt->gm);
                        if(Netpt->constant_db) SET
                        set_generator_load(ODpt->load,poisson(xflow));
                        }
                }

        else if(p[imax-1]<=ran&&ran<p[imax])    /*there is an arrival*/
                {
                if(mode==1)     /*if metering find desired path*/
                        {
                        i_d=0;
                        mindif = ODpt->k[0] - ODpt->kd[0];
                        for(i=1;i<imax;++i)
                                {
```

```
                        if (ODpt->kd[i] <= 0.4) goto find;
                        dif = ODpt->k[i] - ODpt->kd[i];
                        if (mindif - dif >=1.0)
                                {
                                i_d = i;
                                mindif = dif;
                                }
                        else if ((dif - mindif < 1.0) &&
                                (ODpt->kd[i] > ODpt->kd[i_d]))
                                {
                                i_d = i;
                                mindif = dif;
                                }
find:                   ;/*continue to find i_d*/
                        }
                }

        else if(mode == 4)/*if metering (4) find desired path*/
                {
                i_d=0;
                if(ODpt->kd[0] <= 0.01)
                        mindif = 1000.0;
                else
                        mindif = ODpt->k[0] / ODpt->kd[0];
                for(i=1;i<imax;++i)
                        {
                        if(ODpt->kd[i] <= 0.01)
                                dif = 1000.0;
                        else
                                dif = ODpt->k[i] / ODpt->kd[i];
                        if (mindif - dif > 0.0)
                                {
                                i_d = i;
                                mindif = dif;
                                }
                        }
                }

        else if (mode == 2)     /*if randomizing the routing*/
                {
                pp = 0.0;
                imax1 = imax - 1;
                for(i=0;i<imax;++i)
                        {
                        pp += ODpt->kd[i];
                        }
                NONZERO p1[0]=ODpt->kd[imax1]/pp;
                else
                        {
                        i_d = 0;
                        goto add;
                        }
                for(i=1;i<imax;++i)
```

```
                                        {
                                        p1[i]=p1[i-1]+(ODpt->kd[imax1-i])/pp;
                                        }

                                if(ran1<p1[0])  /*path[imax1] has an arrival*/
                                        {
                                        i_d = imax1;
                                        goto add;
                                        }

                                for(i=0;i<(imax-1);++i) /*path[imax1-1-i]has a arrival*/
                                        {
                                        if((p1[i]<=ran1)&&(ran1<p1[i+1]))
                                                {
                                                i_d = imax1 - i - 1;
                                                goto add;
                                                }
                                        }
                                }

                        else if (mode==3)       /*if shortest path routing*/
                                {
                                i_d=ODpt->i_sh;
                                }
add:
                        xflow = (ktot + 1.0)*(ODpt->gm);
                        ODpt->k[i_d]=ODpt->k[i_d]+1.0;
                        if(Netpt->constant_db) SET
                        set_generator_load(ODpt->load,poisson(xflow));
                        }

                else for(i=0;i<(imax-1);++i)            /*path[i+1] has a departure*/
                       {
                        if(p[i]<=ran&&ran<p[i+1])
                        {
                        if(ODpt->k[i+1]>=1.0)
                                {
                                ODpt->k[i+1]=ODpt->k[i+1]-1.0;
                                xflow = (ktot - 1.0)*(ODpt->gm);
                                if(Netpt->constant_db) SET
                                set_generator_load(ODpt->load,poisson(xflow));
                                }
                        }
                       }

prin:   ;       /*if need to print some data*/
        if(ODpt->update_flag)
                {
                printOD(ndpt,ODpt);
                }
        else if(Netpt->print_flag)
                {
                printOD(ndpt,ODpt);
                }
        }

        /*Routine to re-route VCs*/

re_route(ODpt)
        OD *ODpt;
        {
        double pp,rand1(),ran,ran1,xflow,ktot,dif,mindif;
        double p,p1;
        int i,imax,i_d,mode;            /*i_d the path to add a VC*/

        /*calculate prob.  so that ran can be tested*/

        mode = Netpt->mode;
        imax=ODpt->no_path;

        ran = rand1();
        ran1 = rand1();
        pp = ODpt->p2;
        p = 0.0;
        ktot = 0.0;

        for(i=0;i<imax;++i)
                {
                ktot += ODpt->k[i];
                p += ODpt->k[i]*pp;
                }
        if(Netpt->det) ktot = ODpt->p1/(ODpt->p2);

        if(Netpt->det) goto reset;

        p1 = p + ODpt->p1;
        if(p1 >1)
                printf("probability too large\n");

        /*test which case:  updating VC nos.*/

        if(ran < p)                     /*there is a departure of VC*/
                {
                if(ktot >= 1.0)
                        {
                        xflow = (ktot - 1.0)*(ODpt->gm);
                        ktot -= 1.0;
                        if(Netpt->constant_db) SET
                        set_generator_load(ODpt->load,poisson(xflow));
                        }
                }

        else if(p <= ran && ran < p1)   /*there is an arrival*/
                {
                xflow = (ktot + 1.0)*(ODpt->gm);
                ktot += 1.0;
                if(Netpt->constant_db) SET
                set_generator_load(ODpt->load,poisson(xflow));
                }

reset:  /*start re-routing the VCs*/
        for(i=0;i<imax;++i)
                {
                ODpt->k[i] = 0.0;
                }

        if(mode == 3)   /*shortest path routing*/
                {
                ODpt->k[ODpt->i_sh] = ktot;
                }

        else if(mode == 4)      /*metering (4) */
                {
                while(ktot > 0.0)
                        {
                        i_d = 0;
                        if(ODpt->kd[0] <= 0.01)
                                mindif = 1000.0;
                        else
                                mindif = ODpt->k[0] / ODpt->kd[0];
                        for(i=1;i<imax;++i)
                                {
                                if(ODpt->kd[i] <= 0.01)
                                        dif = 1000.0;
                                else
                                        dif = ODpt->k[i] / ODpt->kd[i];
                                if (mindif - dif >  0.0)
                                        {
                                        mindif = dif;
                                        i_d = i;
                                        }
                                }
                        ODpt->k[i_d] += 1.0;
                        ktot -= 1.0;
                        }
                }

        else if(mode == 1)      /*metering*/
                {
                while(ktot > 0.0)
                        {
                        i_d = 0;
                        mindif = ODpt->k[0]-ODpt->kd[0];
                        for(i=1;i<imax;++i)
                                {
                                if (ODpt->kd[i] <= 0.4) goto find;
                                dif = ODpt->k[i]-ODpt->kd[i];
                                if (mindif - dif >= 1.0)
                                        {
```

```
                                        mindif = dif;
                                        i_d = i;
                                        }
                                else if ((dif - mindif < 1.0) &&
                                        (ODpt->kd[i] > ODpt->kd[i_d]))
                                        {
                                        i_d = i;
                                        mindif = dif;
                                        }
        find:                           ;
                                }
                        ODpt->k[i_d] += 1.0;
                        ktot -= 1.0;
                        }
                }

         else printf("re-routing dose not allow randomization\n");
        }

        /*This routine updates flows for the overall Network*/

Net_update()
        {
        int i,imax,j,t,i_first,moving_a;
        double x,beta;

        moving_a = Netpt->moving_a;
        imax = Netpt->linktot;
        if (moving_a)           /*if moving average*/
                {
                t = Netpt->update_int;
                for(i=0;i<imax;++i)
                        {
                        x = get_external(Netpt->link_name[i])/t;
                        i_first = Netpt->i_first[i];

                        if(t == 1)
                                {
                                Netpt->link_f[i] = x*t;
                                goto repeat;
                                }

                        Netpt->link_f[i] -= Netpt->link_ft[i][i_first];
                        Netpt->link_ft[i][i_first] = x;
                        Netpt->link_f[i] += x;
                        Netpt->i_first[i] = (i_first + 1)%t;
repeat:                 ;
                        }
                }
        else
                {
                beta = Netpt->beta;
                for (i=0;i<imax;++i)
```

```
                        {
                        x = get_external(Netpt->link_name[i]);
                        Netpt->link_f[i] = beta*x + (1-beta)*(Netpt->link_f[i]);
                        }
                }

        }

        /*This routine updates ODpair path flows*/

OD_update(ODpt)
        OD *ODpt;
        {
        int i,imax,t,j,i_first,moving_a;
        double x,beta;

        /*update short-term path flows*/

        moving_a = Netpt->moving_a;
        imax = ODpt->no_path;
        t = ODpt->p_update_int;

        if (moving_a)           /*if moving average*/
                {
                if(t == 1)
                        {
                        for(i=0;i<imax;++i)
                                ODpt->x[i] = ODpt->xbuf[i];
                        return;
                        }

                for(i=0;i<imax;++i)
                        {
                        x = ODpt->xbuf[i]/t;
                        i_first = ODpt->i_first[i];

                        ODpt->x[i] -= ODpt->xt[i][i_first];
                        ODpt->xt[i][i_first] = x;
                        ODpt->x[i] += x;
                        ODpt->i_first[i] = (i_first + 1)%t;
                        }
                }
        else
                {
                beta = Netpt->beta;
                for(i=0;i<imax;++i)
                        {
                        x = ODpt->xbuf[i];
                        ODpt->x[i] = beta*x + (1-beta)*(ODpt->x[i]);
                        }
                }

        }

        /*This routine updates the states of a node*/

nd_update(ndpt)
        node *ndpt;
        {
        int i,imax,k1,k2;

        /*First update the link flows for the links locally attached*/
        /*if instantaneous info.  is available, update all flows*/

        if(Netpt->instant)
                {
                imax = Netpt->linktot;
                for(i=0;i<imax;++i)
                        {
                        ndpt->ntpt->link_f[i] = Netpt->link_f[i];
                        ndpt->ntpt->time[i] = Time;
                        }
                goto ODupdate;
                }

        imax=ndpt->no_link;
        for(i=0;i<imax;++i)
                {
                k1 = ndpt->link_no[i][0];
                k2 = ndpt->link_no[i][1];
                ndpt->ntpt->link_f[k1]=Netpt->link_f[k1];
                ndpt->ntpt->link_f[k2]=Netpt->link_f[k2];
                ndpt->ntpt->time[k1] = Time;
                ndpt->ntpt->time[k2] = Time;
                }

        /* perform OD_update, r_update,dynrt for OD pairs */

ODupdate:
        imax = ndpt->no_OD;
        for(i=0;i<imax;++i)
                {
                OD_update(ndpt->ODpt[i]);
                r_update(ndpt,ndpt->ODpt[i]);
                dynrt(ndpt,ndpt->ODpt[i]);
                }
        }
/*                          File update.c
This file contains the routing controller and other supporting routines.
r_update (the routing controller):  updates the routing variables;
Dcal    :calculate the cost;
printNet:print the state of network;
printOD :print the state of an OD pair;
OD_printNet:  print the table of the estimates of link flows available locally.*/

#include <stdio.h>
```

```
#include <math.h>
#include <opnet.h>
#include "struct.h"

extern FILE *fout;
extern Net *Netpt;

        /*This function calculate costs*/

Dcal()
        {
        double d,f,c,ff;
        int i,imax;
        imax=Netpt->linktot;
        d=0.0;
        for(i=0;i<imax;++i)
                {
                f = Netpt->link_f[i];
                c = Netpt->c[i];
                if(f > 0.99*c)
                        {
                        ff = f - 0.99*c;
                        ff = 99.0 + 10000.0*ff/c + 1000000.0*ff*ff/(c*c);
                        d += ff;
                        }
                else    d += f/(c-f);
                }
        cont_save_variable("cost",d);
        Netpt->cost = d;
        }

        /*This function prints Network flows*/

printNet()
        {
        double d,f,c,ff,g,de;
        int i,imax;
        imax = Netpt->linktot;
        d = 0.0;
        fprintf(fout,"\n**** Time = %d Network-wide ****\n\n",Time);
        for(i=0;i<imax;++i)
                {
                f = Netpt->link_f[i];
                c = Netpt->c[i];
                g = f/c;
                if(f > 0.99*c)
                        {
                        ff = f - 0.99*c;
                        ff = 99.0 + 10000.0*ff/c + 1000000.0*ff*ff/(c*c);
                        d += ff;
                        de = ff;
                        }
                else
```

```
                        {
                        de = f/(c-f);
                        d += de;
                        }
                fprintf(fout,"f[%d] = %f f/c[%d] = %f d[%d] = %f\n",
                        i,Netpt->link_f[i],i,g,i,de);
                }

        fprintf(fout,"cost = %f\n",d);
        }

OD_printNet(ndpt,ODpt)
        OD *ODpt;
        node *ndpt;
        {
        int i,imax;

        imax=Netpt->linktot;
        fprintf(fout,"\n**** Time = %d OD pair %s ****\n\n",Time,ODpt->name);
        for(i=0;i<imax;++i)
                {
                fprintf(fout,"link flow [%d] = %f time[%d] = %d\n",
                i,ndpt->ntpt->link_f[i],i,ndpt->ntpt->time[i]);
                }

        }

        /*This function prints OD flows*/

printOD(ndpt,ODpt)
        OD *ODpt;
        node *ndpt;
        {
        int i,imax;
        double k,kd,x,xd;

        imax=ODpt->no_path;

/*      OD_printNet(ndpt,ODpt); */
        fprintf(fout,"\n**** Time = %d OD pair %s ****\n\n",Time,ODpt->name);
        fprintf(fout,"i_sh = %d\n\n",ODpt->i_sh);

        for(i=0;i<imax;++i)
                {
                k=ODpt->k[i];
                kd=ODpt->kd[i];
                fprintf(fout,"k[%d]= %f kd[%d]= %f\n",i,k,i,kd);
                }

        for(i=0;i<imax;++i)
                {
                x=ODpt->x[i];
                xd=ODpt->xd[i];
                fprintf(fout,"x[%d]= %f xd[%d]= %f\n",i,x,i,xd);
                }
        }

        /*This function updates routing variables*/

r_update(ndpt,ODpt)
        OD *ODpt;
        node *ndpt;
        {
        int i_sh,t,synch;               /*the shortest path is i_sh*/
        int i,imax,j,jmax,k,kmax,diff;
        double d,d1,xnonsh,xtot,x1,f,c,ff;
        double pd[maxp],pd1[maxp];      /*the FDL and SDL of paths*/
        double ld2[maxltot],ld1[maxltot];       /*the link FDL and SDL*/

        synch = Netpt->synch;
        if(synch)       /*if synchronous routing*/
                {
                t = ODpt->r_update_time;
                if(Time!=t) return;
                goto start;
                }
        diff = Time - ODpt->old_update_t;
        if((ndpt->update <= 0) && (diff < Netpt->tl)) return;

        /*Calculate Link FDL & SDL*/
start:
        imax = Netpt->linktot;
        for(i=0;i<imax;++i)
                {
                f = ndpt->ntpt->link_f[i];
                c = Netpt->c[i];
                if(f > 0.99*c)
                        {
                        ld1[i] = 10000.0/c;
                        ld2[i] = 2000000.0/(c*c);
                        }
                else
                        {
                        ff = (c-f)*(c-f);
                        ld1[i] = c/ff;
                        ld2[i] = 2*c/(ff*(c-f));
                        }
                }

        /*claculate FDL*/

        imax=ODpt->no_path;
        for(i=0;i<imax;++i)
                {
                d=0.0;
                d1=0.0;
                jmax=ODpt->path[i].no_link;
                for(j=0;j<jmax;++j)
                        {
                        k = ODpt->path[i].link_no[j];   /*link no.*/
                        d += ld1[k];
                        d1 += ld2[k];
                        }
                pd[i] = d;
                pd1[i] = d1;
                }

        /*find the shortest path i_sh*/

        i_sh = 0;
        for(i=0;i<imax;++i)
                {
                if(pd[i]<pd[i_sh])
                        i_sh=i;
                }

        /*calculate current total flow:  xtot*/

        xtot = 0.0;
        for(i=0;i<imax;++i)
                {
                xtot += ODpt->x[i];
                }

        if(synch) goto proceed1;

        /*proceed to r_update*/

proceed:                /*I am now committed to asynchronous r_update*/

        ndpt->update -= 1;

        ODpt->old_update_t = Time;

proceed1:               /*I am now committed to synchronous r_update*/

        ODpt->i_sh= i_sh;
        if(Netpt->mode == 3) /*if shortest path routing*/
                goto end_update;

        if(Netpt->print_flag)
                {
                fprintf(fout,"\n### Routing Update Time = %d ###\n",Time);
                fprintf(fout," OD pair %s\n\n",ODpt->name);
                ODpt->update_flag=1;
                for(i=0;i<imax;++i)
                        {
                        fprintf(fout,"d[%d]= %f d_1[%d]= %f\n",
                                i,pd[i],i,pd1[i]);
                        }
                }

        /*calculate SDL*/

        for(i=0;i<imax;++i)
                if(i!=i_sh)
                        {
                        pd1[i] += pd1[i_sh];
                        jmax=ODpt->path[i].no_link;
                        kmax=ODpt->path[i_sh].no_link;
                        for(j=0;j<jmax;++j)
                                {
                                for(k=0;k<kmax;++k)
                                        {
                 if(ODpt->path[i].link_no[j]==ODpt->path[i_sh].link_no[k])
                        pd1[i] -= ld2[ODpt->path[i].link_no[j]]*2.0;
                                        }
                                }
                        }


        /*claculate desired flows*/

        d = pd[i_sh];                   /*the shortest FDL*/
        xnonsh = 0.0;
        x1=(ODpt->gm)*(ODpt->dbsize);   /*one unit of bitflow per VC*/

        for(i=0;i<imax;++i)
                {
                if(i!=i_sh)
                        {
                        d1 = pd1[i];
                        if(Netpt->algo == 1)
                                {
                                ODpt->xd[i]=fmax(0.0,
                                        ODpt->x[i]-(Netpt->af)*(pd[i]-d)/d1);
                                }
                        else if(Netpt->algo == 2)
                                {
                                ODpt->xd[i]=fmax(0.0,
                                        ODpt->xd[i]-(Netpt->af)*(pd[i]-d)/d1);
                                }
                        ODpt->kd[i]=ODpt->xd[i]/x1;
                        xnonsh=xnonsh+ODpt->xd[i];
                        }
                }

        ODpt->xd[i_sh] = xtot - xnonsh;
        ODpt->kd[i_sh] = ODpt->xd[i_sh] / x1;
end_update:
        ;
        if(synch)
```

```
                ODpt->r_update_time += ODpt->r_update_T;/*set next update time*/

        }
/*                      File proc.c
This file contains the routing processor, routing-server,
and the link-server, and other supporting routines.
Packets are referred to as Data-Blocks, or simply dbs.
routing_server:  processes data-blocks for a node, tasks including
                sending data-blocks for VCs originating locally,
                reading the contol data-blocks,
                flooding control data-blocks,
                and continuing a flooding by copying cotrol data-blocks and
                sending the copies to neighbors;
send_db:  sends useful control data-blocks away;
set_path:  sends data-blocks for the VCs originating locally;
check_db:  checks to see if the received control db is useful;
cpoy_cdb:  copies a useful control data-block;
link_server:  makes sure the bit rate transmitted is not greater than
                the pre-specified rate for the transmitter attached.*/

#include <opnet.h>
#include <stdio.h>
#include <math.h>
#include "struct.h"

extern Net *Netpt;

/*Field description of a data-block
        field 0 :       path number :0 => control db
        field 1 :       link number
        field 2 :       time stamp
        field 3 :       link flow
        field 4 :       link to be traversed
                                */

        /*routine origins control db*/

send_db(lno,ndpt,outch,outlink,dbsize,orig)
        int lno,dbsize,outch,outlink,orig;
        node *ndpt;
        {
        Data_Block *dbpt;

        dbpt = proc_create_db(dbsize);
        proc_set_field(dbpt,0,0.0,0);                   /*assign path no*/
        proc_set_field(dbpt,1,((double) lno),0);        /*assign link no*/
        proc_set_field(dbpt,2,((double) Time),0);       /*assign time stamp*/
        proc_set_field(dbpt,4,((double) outlink),0);    /*assign outlink*/
        proc_set_field(dbpt,3,ndpt->ntpt->link_f[lno],0);/*assign linkflow*/
        proc_output_db(dbpt,outch);
        }

        /*routine copies control db*/
```

```

copy_cdb(dbpt,outch,outlink,orig)
        int outch,outlink,orig;
        Data_Block *dbpt;
        {
        Data_Block *dbpt1;

        dbpt1 = proc_copy_db(dbpt);
        proc_set_field(dbpt1,4,((double) outlink),0);   /*assign outlink*/
        proc_output_db(dbpt1,outch);

        }

        /*routine reads control db*/

check_db(dbpt,pno,ndpt,dest)
        node *ndpt;
        Data_Block *dbpt;
        int dest,pno;
        {
        int lno,time;
        lno = (int) proc_get_field(dbpt,1);
        time = (int) proc_get_field(dbpt,2);
        if(time > ndpt->ntpt->time[lno])
                {
                ndpt->ntpt->link_f[lno] = proc_get_field(dbpt,3);
                ndpt->ntpt->time[lno] = time;

                ndpt->update = 2;
                return (1);             /*if useful control db read*/
                }
        else
                {
                proc_destroy_db(dbpt);
                return (0);             /*if redundant info*/
                }
        }

                /*routine to set path numbers and routs them*/

set_path(ODpt)
        OD *ODpt;
        {
        int     i,imax,i_d,inch,pno,outch,k,kmax,j,jmax,db_no = 0;
        int     bitsize;
        double  pp,ran,rand1();
        double  p[maxp];
        Data_Block      *dbpt;

        inch = ODpt->set_path_inch;
        imax = ODpt->no_path;
        pp = 0.0;
        for(i=0;i<imax;++i)
```

```
                {
                pp += ODpt->k[i];
                ODpt->xbuf[i] = 0.0;
                }
        if(pp == 0.0) return;

        if(Netpt->constant_db)  goto constant;

        p[0] = ODpt->k[0]/pp;
        for(i=1;i<imax;++i)
                p[i] = p[i-1] + (ODpt->k[i])/pp;

        while(!proc_is_stream_empty(inch))
                {
                dbpt = proc_get_first(inch);
                ran = rand1();

                if(ran < p[0])  /*if path[0] has an arrival*/
                        {
                        i_d = 0;
                        }
                else for(i=0;i<(imax-1);++i)    /*if path[i+1] has an arrival*/
                        {
                        if(p[i] <= ran && ran <p[i+1])
                                {
                                i_d = i+1;
                                goto setting;
                                }
                        }
setting:
                pno = ODpt->set_path_pno[i_d];
                outch = ODpt->set_path_outch[i_d];
                bitsize = proc_get_bitcount(dbpt);

                proc_set_field (dbpt,0,((double) pno),0);
                ++db_no;
                proc_output_db(dbpt,outch);
                ODpt->xbuf[i_d] += (double) bitsize;
                }

        for (i=0;i<imax;++i)
                Netpt->load += ODpt->xbuf[i];

/*      printf("OD name :  %s db marked = %d Time = %d\n",
                ODpt->name,db_no,Time);         */
        return;

constant:       /*if the VC number is constant, and the packet rate per
                VC is constant, we have to send the data-blocks differently*/

        jmax = (int) ODpt->gm;
        for(i=0;i<imax;++i)
                {
```

```
                kmax = (int) (0.5 + ODpt->k[i]);
                for(k=0;k<kmax;++k)
                        {
                        for(j=0;j<jmax;++j)
                                {
                                dbpt = proc_get_first(inch);
                                pno = ODpt->set_path_pno[i];
                                outch = ODpt->set_path_outch[i];

                                proc_set_field (dbpt,0,((double) pno),0);
                                ++db_no;
                                proc_output_db(dbpt,outch);
                                ODpt->xbuf[i] += (double)
                                        proc_get_bitcount(dbpt);
                                }
                        }
                }
/*      printf("OD name :  %s db marked = %d Time = %d\n",
                ODpt->name,db_no,Time);         */
        }

        /*This is a server attached to a link with capacity cp*/

link_server(cp)
        int cp;
        {
        int     active,total = 0;
        int     bitcount,diff,cbitsize;
        Queue   *quept;
        Data_Block *dbpt,*held_dbpt;

        if(Time == 0)   /*initialize the state variable*/
                {
                proc_alloc_state(sizeof(state_var));
                ((state_var*) proc_state_ptr())->active = 0;
                }

        quept = proc_get_queue_ptr(0);          /*the queue pointer*/
        cbitsize = QueueSLE(quept,0,0.0,cp);    /*the bandwidth used by
                                                control dbs*/

        cp -= cbitsize;
        proc_output_stream(0,0);
        if(cp == 0)     return;/*if the control dbs use up all the bandwidth*/

        /*assess the state to see if there is a left over db to be sent*/

        held_dbpt = &(((state_var*) proc_state_ptr())->held_db);
        active = ((state_var*) proc_state_ptr())->active;

        if(active)      /*if there is a left over db to be sent*/
                {
                dbpt = proc_copy_db(held_dbpt);
                bitcount = proc_get_bitcount(dbpt);
                diff = bitcount - cp;
                if(diff > 0)/*if the left over db has to be brokened up*/
                        {
                        proc_set_bitcount(dbpt,cp);
                        proc_set_bitcount(held_dbpt,diff);
                        proc_output_db(dbpt,0);
                        return;
                        }
                else if(diff == 0)/*if the left over db ueses up all the
                                        available bandwidth*/
                        {
                        proc_output_db(dbpt,0);
                        ((state_var*) proc_state_ptr())->active = 0;
                        return;
                        }

                /*Now there are some left over bandwidth to send more dbs*/

                proc_output_db(dbpt,0);
                total += bitcount;
                ((state_var*) proc_state_ptr())->active = 0;
                }

repeat: /*start sending data packets*/
        proc_access_head(0);
        if(proc_is_stream_empty(0))
                {
                return;
                }
        dbpt = proc_get_first(0);
        bitcount = proc_get_bitcount(dbpt);
        diff = total + bitcount - cp;
        if(diff < 0)    /*more bandwidth available*/
                {
                proc_output_db(dbpt,0);
                total += bitcount;
                goto repeat;
                }
        else if (diff == 0)/*the bandwidth is just exhausted*/
                {
                proc_output_db(dbpt,0);
                return;
                }
        else            /*a db must be brokened up and left over*/
                {
                proc_replicate_db(dbpt,held_dbpt);
                proc_set_bitcount(held_dbpt,diff);
                proc_set_bitcount(dbpt,(cp - total));
                proc_output_db(dbpt,0);
                ((state_var*) proc_state_ptr())->active = 1;
                return;
                }

        }

        /*This is a general purpose routing server*/

routing_server(ndpt,RTpt)
        RT *RTpt;
        node *ndpt;
        {
        int i,imax,j,jmax,T,node_no,outlink,outch,lno1,lno2,pno,synch;
        int tm,ts,diff;
        double fabs(),f1,f2,f3;
        Data_Block *dbpt;
        int db_no = 0;

        node_no = RTpt->node_no;
        imax = ndpt->no_OD;
        for(i=0;i<imax;++i)     /*sending the dbs for the local VCs*/
                set_path(ndpt->ODpt[i]);

        imax = RTpt->no_receiver;/*checks the dbs received*/
        for(i=1;i<=imax;++i)
                {
                while(!proc_is_stream_empty(i))
                        {
                        dbpt = proc_get_first(i);
                        pno = (int) proc_get_field(dbpt,0);
                        if(pno != 0)
                                {
                                jmax = RTpt->no_path;
                                /*route the dbs*/
                                for(j=0;j<jmax;++j)
                                        {
                                        if(pno == RTpt->path_no[j])
                                                {
                                                outch = RTpt->outch[j];
                                                if(outch < 0)
                                                        {
                                                        proc_destroy_db(dbpt);
                                                        ++db_no;
                                                        goto end_route;
                                                        }
                                                proc_output_db(dbpt,outch);
                                                goto end_route;
                                                }
                                        }
                                printf("db not routed pno = %d\n",pno);
                                printf("node_no = %d\n",node_no);
        end_route:                      ;
                                }
                        /*if need to flood*/
                        else if(check_db(dbpt,pno,ndpt,node_no))
                                {
```

```
                                jmax = ndpt->no_link;
                                lno1 = (int) proc_get_field(dbpt,4);
                                for(j=0;j<jmax;++j)
                                        {
                                        if(ndpt->link_no[j][0] != lno1)
                                          {
                                          outch = ndpt->link_no[j][2];
                                          outlink = ndpt->link_no[j][1];
                                          copy_cdb(dbpt,outch,outlink,node_no);
                                          }
                                        }
                                proc_destroy_db(dbpt);
                                }
                        }
                }

/*      printf("Node no:  %d db destroyed = %d Time = %d\n",node_no,
                                db_no,Time);            */

                /*send the control db*/

        if(Netpt->instant) return;      /*if instant send no control db*/

        synch = Netpt->synch;
        if(synch)
                {
                T = ndpt->send_int;
                if(Time < T) return;
                if(Time%T == ndpt->send_bias)
                        {
                        imax = ndpt->no_link;
                        jmax = imax;
                        for(i=0;i<imax;++i)
                                {
                                lno2 = ndpt->link_no[i][1];
                                for(j=0;j<jmax;++j)
                                        {
                                        if(i != j)
                                          {
                                          outlink = ndpt->link_no[j][1];
                                          outch = ndpt->link_no[j][2];
                                          send_db(lno2,ndpt,outch,outlink,
                                                Netpt->c_dbsize,node_no);
                                          }
                                        }
                                }
                        }
                }
        else
                {
                tm = Netpt->tm;
                imax = ndpt->no_link;
                jmax = imax;
                for(i=0;i<imax;++i)
                        {
                        lno2 = ndpt->link_no[i][1];

                        /*test if need to flood*/

                        diff = Time - Netpt->old_flood_t[lno2];
                        if(diff <= tm) goto repeat;
                        f1 = Netpt->link_f[lno2]/Netpt->c[lno2];
                        f3 = Netpt->link_fo[lno2];
                        f2 = fabs(f1 - f3);

                        if(f1 > Netpt->m_thre)
                                {
                                goto flood;
                                }
                        else
                                {
                                if(f2 >= Netpt->diff[lno2]) goto flood;
                                Netpt->diff[lno2] -= Netpt->small;
                                goto repeat;
                                }

                        /*start flooding*/
flood:
                        ndpt->update = 2;
                        Netpt->link_fo[lno2] = f1;
                        Netpt->old_flood_t[lno2] = Time;
                        Netpt->diff[lno2] = Netpt->f_thre;

                        for(j=0;j<jmax;++j)
                                {
                                if(i != j)
                                  {
                                  outlink = ndpt->link_no[j][1];
                                  outch = ndpt->link_no[j][2];
                                  send_db(lno2,ndpt,outch,outlink,
                                        Netpt->c_dbsize,node_no);
                                  }
                                }
repeat:                 ;/*continue to see if other links attcahed
                                need flooding*/
                        }
                }
        }
/*                      File q.c
This file contains the routine implementing the priority queueing.
QueueSLE is a Queue-access method which gives control db full priority;
QueueLE is a Queue-access method which make sure that specify bandwidth
is not exceeded.  */

#include <opnet.h>

```

```
/* Queue-access method:  select dbs whose field(fieldnum)=val, w/ total bandwd */

QueueSLE(quept,fieldnum,val,bandwd)
        Queue*  quept;
        int fieldnum,bandwd;
        double val;
        {
        int total = 0;
        int bitcount;

        queue_set_pointer(quept);

        if(queue_is_empty()) return 0;
        queue_goto_head();
        while(!queue_is_finished())
                {
                if(queue_db_field(fieldnum) == val)
                        {
                        bitcount = queue_db_bitcount();
                        if((total + bitcount) <= bandwd)
                                {
                                queue_pop();
                                total += bitcount;
                                }
                        else    return (total);
                        }
                else    queue_backward();
                }
        return (total);
        }

/* Queue-access method:  w/ total bandwd */

QueueLE(quept,bandwd)
        Queue*  quept;
        int bandwd;
        {
        int diff,total = 0;
        int bitcount;
        Data_Block *dbpt;

        queue_set_pointer(quept);

        if(queue_is_empty()) return 0;
        queue_goto_head();
        while(!queue_is_finished())
                {
                bitcount = queue_db_bitcount();
                diff = total + bitcount - bandwd;
                if(diff < 0)
                        {
                        queue_pop();
                        total += bitcount;
                        }
                else if(diff == 0)
                        {
                        queue_pop();
                        return (bandwd);
                        }
                else
                        {
                        dbpt = queue_copy_db(bandwd - total);/*cp w/ bitcount*/
                        queue_set_bitcount(diff);       /*set head bitcount*/
                        queue_load(dbpt);       /*load the created db*/
                        return (bandwd);
                        }
                }
        return (total);
        }
```

```
/*                      File cnet8.c
This file contains routines calling the controller codes*/

#include <stdio.h>
#include <opnet.h>
#include "struct.h"

Net Netwk;
Net *Netpt;
OD w13,w24,w35,w46,w57,w68,w71,w82,w17,w28,w31,w42,w53,w64,w75,w86;

node n1,n2,n3,n4,n5,n6,n7,n8;
ntcp nt1,nt2,nt3,nt4,nt5,nt6,nt7,nt8;
node *n1pt = &n1;
node *n2pt = &n2;
node *n3pt = &n3;
node *n4pt = &n4;
node *n5pt = &n5;
node *n6pt = &n6;
node *n7pt = &n7;
node *n8pt = &n8;

RT RT1,RT2,RT3,RT4,RT5,RT6,RT7,RT8;
FILE *fout;
double avg=0.0;

        /*controller ct2*/

ct2()
        {
        NCC(51,&Netwk,&n2,"fnet8","fn2",&w24,&w28,&nt2,&RT2);
        if(Time >= 3000)
                avg += Netpt->cost/4000.0;
        if(Time == 7000)
                cont_save_variable("avg",avg);
        if(Time == 7000 )
                {
```

```
                printOD(n1pt,&w13);
                printOD(n1pt,&w17);
                printOD(n2pt,&w24);
                printOD(n2pt,&w28);
                printOD(n3pt,&w35);
                printOD(n3pt,&w31);
                printOD(n4pt,&w46);
                printOD(n4pt,&w42);
                printOD(n5pt,&w57);
                printOD(n5pt,&w53);
                printOD(n6pt,&w68);
                printOD(n6pt,&w64);
                printOD(n7pt,&w71);
                printOD(n7pt,&w75);
                printOD(n8pt,&w82);
                printOD(n8pt,&w86);
                fprintf(fout,"\n cost = %f\n",Netpt->cost);
                }
        }

        /*controller ct1*/

ct1()
        {
        cn(&n5,"fn5",&w57,&w53,&nt5,&RT5);
        }

        /*controller ct3*/

ct3()
        {
        if(cont_dev_id() == 1) cn(&n3,"fn3",&w31,&w35,&nt3,&RT3);
        else if(cont_dev_id() == 2) cn(&n4,"fn4",&w46,&w42,&nt4,&RT4);
        else if(cont_dev_id() == 4) cn(&n6,"fn6",&w68,&w64,&nt6,&RT6);
        else if(cont_dev_id() == 5) cn(&n1,"fn1",&w13,&w17,&nt1,&RT1);
        }

        /*controller ct4*/

ct4()
        {
        if(cont_dev_id() == 6) cn(&n7,"fn7",&w71,&w75,&nt7,&RT7);
        else if(cont_dev_id() == 7) cn(&n8,"fn8",&w82,&w86,&nt8,&RT8);
        }
/*                      File pnet8.c
This file contains routines calling the processor codes*/

#include <opnet.h>
#include "struct.h"

extern Net *Netpt;
extern node *n1pt,*n2pt,*n3pt,*n4pt,*n5pt,*n6pt,*n7pt,*n8pt;

```

```
        /*server st1*/

st1()
        {
        routing_server(n5pt,n5pt->RTpt);
        }

        /*server st2*/

st2()
        {
        Dcal();
        routing_server(n2pt,n2pt->RTpt);
        }

        /*server st3*/

st3()
        {
        if(proc_dev_id() == 1)  routing_server(n3pt,n3pt->RTpt);
        else if(proc_dev_id() == 2)     routing_server(n4pt,n4pt->RTpt);
        else if(proc_dev_id() == 4)     routing_server(n6pt,n6pt->RTpt);
        else if(proc_dev_id() == 5)     routing_server(n1pt,n1pt->RTpt);
        }

        /*server st4*/

st4()
        {
        if(proc_dev_id() == 6)  routing_server(n7pt,n7pt->RTpt);
        else if(proc_dev_id() == 7)
                {
                routing_server(n8pt,n8pt->RTpt);
/*              proc_save_variable("load",Netpt->load);         */
                }
        }

        /*server s500*/

s500()
        {
        link_server(50);
        }

        /*server s1000*/

s1000()
        {
        link_server(100);
        }
```