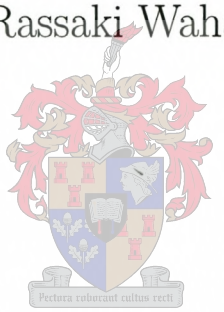
Resource Management in IP Networks

AbdoulRassaki Wahabi



Thesis presented in partial fulfilment
of the requirements for the degree of
Master of Science
at The University of Stellenbosch

Supervisor: Prof. A. E. Krzesinski

December 2001

Declaration

I, the undersigned, hereby declare that the work contained in this thesis is my own original work and has not previously in its entirety or in part been submitted at any university for a degree.

Abstract

IP networks offer scalability and flexibility for rapid deployment of value added IP services. However, with the increased demand and explosive growth of the Internet, carriers require a network infrastructure that is dependable, predictable, and offers consistent network performance.

This thesis examines the functionality, performance and implementation aspects of the MPLS mechanisms to minimize the expected packet delay in MPLS networks. Optimal path selection and the assignment of bandwidth to those paths for minimizing the average packet delay are investigated.

We present an efficient flow deviation algorithm (EFDA) which assigns a small amount of flow from a set of routes connecting each OD pair to the shortest path connecting the OD pair in the network. The flow is assigned in such a way that the network average packet delay is minimized. Bellman's algorithm is used to find the shortest routes between all OD pairs. The thesis studies the problem of determining the routes between an OD pair and assigning capacities to those routes.

The EFDA algorithm iteratively determines the global minimum of the objective function. We also use the optimal flows to compute the optimal link capacities in both single and multirate networks. The algorithm has been applied to several examples and to different models of networks. The results are used to evaluate the performance of the EFDA algorithm and compare the optimal solutions obtained with different starting topologies and different techniques. They all fall within a close cost-performance range. They are all within the same range from the optimal solution as well.

Opsomming

IP-netwerke voorsien die skaleerbaarheid en buigsaamheid vir die vinnige ontplooing van toegevoegde-waarde IP-dienste. Die vergrote aanvraag en eksplosiewe uitbreiding van die Internet benodig betroubare, voorspelbare en bestendige netwerkprestasie.

Hierdie tesis ondersoek die funksionaliteit, prestasie en implementering van die MPLS(multi-protokoletiketskakel)-meganismes om die verwagte pakketvertraging te minimeer.

Ons bespreek 'n doeltreffende algoritme vir vloei-afwyking (EFDA) wat 'n klein hoeveelheid vloei toewys uit die versameling van roetes wat elke OT(oorsprong-teiken)-paar verbind aan die kortste pad wat die OT-paar koppel. Die vloei word toegewys sodanig dat die netwerk se gemiddelde pakketvertraging geminimeer word. Bellman se algoritme word gebruik om die kortste roetes tussen alle OT-pare te bepaal. Die tesis bespreek die probleem van die bepaling van roetes tussen 'n OT-paar en die toewysing van kapasiteite aan sulke roetes.

Die EFDA-algoritme bepaal die globale minimum iteratief. Ons gebruik ook optimale vloeie vir die berekening van die optimale skakelkapasiteite in beide enkel- en multikoers netwerke. Die algoritme is toegepas op verskeie voorbeelde en op verskillende netwerkmodelle. Die skakelkapasiteite word aangewend om die prestasie van die EFDA-algoritme te evalueer en dit te vergelyk met die optimale oplossings verkry met verskillende aanvangstopologieë en tegnieke. Die resultate val binne klein koste-prestasie perke wat ook na aan die optimale oplossing lê.

Acknowledgements

Many people have contributed in some or other way to getting this thesis finished. To these people I owe an enormous debt of gratitude.

- Professor Tony Krzesinski for his academic example, wisdom, and continued support for this work.
- My parents (mom and dad) for their multi-faceted support which allowed me to come to South Africa.
- Reg Dodds, for his timely assistance and being a friend.
- The rest of the Computer Science Department academic staff, students and colleagues for their collaboration and friendship.
- The friends I have made over the past few years at Stellenbosch who have all contributed in numerous and immeasurable ways.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

This introductory chapter begins with an overview of resource management in IP networks. The role of optimization of MPLS networks is introduced. Finally, a brief discussion introduces each of the six major parts of this thesis.

1.1 Network Resource Management

In the data communications and telecommunications industries today, there is a broad consensus that the era of circuit-switched networks is drawing to a close. Circuit-switched networks will be gradually replaced with packet-switched networks offering better scalability along with enhanced handing of data traffic. Today's circuit-switching applications - for example, real-time voice - will be mapped to virtual calls across the packet-switched network.

However, this broad consensus is over a decade old, and there have been many announcements of new technologies that promise to change the way data are forwarded, or switched in the Internet or other networks. First Integrated Services Digital Network (ISDN), then Asynchronous Transfer Mode (ATM), and most recently Internet Protocol (IP) have been viewed as traffic technologies that would unite all of the diverse forms of communications once. Many of these technologies are based on a set of common ideas. They all use a label swapping technique for forwarding data.

MPLS - Multiprotocol Label Switching is an approach proposed by the Internet Engineering Task Force (IETF) which promises to play a fundamental role in uniting IP and ATM technology. It is to be the networking technology to deliver traffic engineering

capability and QoS performance for carrier networks.

In an MPLS network, incoming packets are assigned a *label* by a Label Edge Router (LER). Packets are forwarded along a Label Switched Path (LSP) where each Label Switched Router (LSR) makes forwarding decisions based on the contents of the label. At each hop, the LSR removes the existing label and applies a new label which tells the next hop router how to forward the packet.

MPLS can deliver control and performance to IP data packets through the use of Label Switched Paths (LSPs), in particular, with the use of explicitly routed LSPs (ER-LSPs).

One of the most significant application of MPLS is in traffic engineering which refers to the control of traffic flow in a network. Conventional IP traffic is dynamic and hard to predict because the flows are constantly changing and therefore do not necessarily match the network topology that has been put in place. Currently, IP traffic within routed networks is forwarded according to layer 3 shortest or lowest cost path algorithms, regardless of downstream conditions. These algorithms do not account for activities within, or the current state of the network. Traffic engineering via MPLS allows the traffic to be mapped efficiently to current network topologies. By setting up paths through a network to accommodate traffic, MPLS offers control over traffic that traditional routing algorithms cannot. The promise of MPLS is that it will directly integrate connections and predictability into IP networks, thereby simplifying network design and management.

The essential concept behind network resource management is to allocate network resources in such a way as to separate traffic flows according to service characteristics.

In the design process, satisfactory resource utilization and good performance can be difficult to achieve simultaneously due to the multiservice environment and both variability and uncertainty of the traffics offered to the network. To improve the performance, some management actions should be introduced into the network with the aim of dynamically adapting the resource assignment to the current traffic levels. To this end, two main aspects of network management are addressed in this thesis. The first concerns traffic flow assignment in MPLS networks to achieve high performance. Several routing algorithms are compared and routing schemes maximizing network throughput are considered. The second considers the dynamic adaptation of link capacities.

The rest of this thesis is structured as follows. In chapter 2, we present an introduction to the principles of MPLS, and we elaborate a view of its benefits and applications.

Chapter 3 focuses on the general design choices and mechanisms we rely on to support QOS requests. This includes details on the path selection metrics, link state update extensions, and the path selection algorithm itself.

Chapter 4 examines the Flow Deviation Algorithm designed to minimize average network delay. The routing algorithms used here are essentially static flow assignment strategies that distribute the flows in the network to minimize the average packet delay.

In Chapter 5, we compute the optimal link capacities by considering the same criterion used in the flow assignment problem in combination with a blocking model for describing call admission controls in multiservice broadband networks. Traffic of a number of different types requiring different bandwidth allocations is offered to each origin-destination pair. The network manager must implement a call admission control scheme to minimize the packet delay and maximize the throughput earned from the network while maintaining agreed quality of service constraints.

Chapter 6 presents some experimental results. The EFDA algorithm has been applied to various test networks. We use the optimal flows to compute the optimal link capacities and compare them with optimal capacities as computed by another approach. We examine the strengths and weaknesses of each and consider the environments in which one approach or the other might be most suitable.

Finally, an appendix provides additional material of interest, namely an equation to compute the Multiservice blocking probability.

Chapter 2

MPLS Networks

This chapter presents an overview of Multiprotocol Label Switching (MPLS), highlighting MPLS in ATM networks and packet-based networks. It concentrates on the fundamentals of MPLS network design that apply to all networks, including those mechanisms supporting traffic engineering.

2.1 Feature Overview

As a packet of a connectionless network layer protocol travels from one router to the next, each router makes an independent forwarding decision for that packet. That is, each router analyzes the packet's header, and each router runs a network layer routing algorithm. Each router independently chooses a next hop for the packet, based on its analysis of the packet's header and the results of running the routing algorithm.

The main concept of MPLS is to introduce a *label* into each packet. Each MPLS packet has a header. The header is between the IP header and the link layer header. Packet headers contain considerably more information than is needed simply to choose the next hop. Choosing the next hop can therefore be thought of as the composition of two functions. The first function separates the entire set of possible packets into a set of *Forwarding Equivalence Classes (FECs)*. The second function maps each FEC to a next hop. All packets which belong to a particular FEC and which travel from a particular node will follow the same path or if certain kinds of multi-path routing are in use, they will all follow one of a set of paths associated with the FEC.

In conventional IP forwarding, a particular router will typically consider two packets

to be in the same FEC if there is some address prefix X in that router's routing tables such that X is the longest match for each packet's destination address. As the packet traverses the network, each hop re-examines the packet header and assigns it to a FEC.

In MPLS, the assignment of a particular packet to a particular FEC is done once, as the packet enters the network. The FEC to which the packet is assigned is encoded as a short fixed length value known as a *label.* When a packet is forwarded to its next hop, the label is sent along with it; that is, the packets are labeled before they are forwarded.

At subsequent hops, there is no further analysis of the packet's network layer header. Rather, the label is used as an index into a table which specifies the next hop, and a new label. The old label is replaced with the new label, and the packet is forwarded to its next hop.

In the MPLS forwarding process, once a packet is assigned to a FEC, no further header analysis is done by subsequent routers; all forwarding is driven by the labels. This has a number of advantages over conventional network layer forwarding.

- MPLS forwarding can be done by routers which are capable of doing label lookup and replacement, but are either not capable of analyzing the network layer headers, or are not capable of analyzing the network layer headers at adequate speed.
- Since a packet is assigned to a FEC when it enters the network, the ingress router may use, in determining the assignment, any information it has about the packet, even if that information cannot be collected from the network layer header. For example, packets arriving on different ports may be assigned to different FECs. Conventional forwarding, on the other hand, can only consider information which travels with the packet in the packet header.
- A packet that enters the network at a particular router can be labeled differently than the same packet entering the network at a different router, and as a result forwarding decisions that depend on the ingress router can be easily made. This cannot be done with conventional forwarding, since the identity of a packet's ingress router does not travel with the packet.
- The considerations that determine how a packet is assigned to a FEC can become complicated, without any impact on the routers that forward labeled packets.
- Sometimes it is desirable to force a packet to follow a particular route which

> is explicitly chosen at or before the time the packet enters the network, rather than being chosen by the normal dynamic routing algorithm as the packet travels through the network. This may be done as a matter of policy or to support traffic engineering. In conventional forwarding, this requires the packet to carry an encoding of its route. In MPLS, a label can be used to represent the route, so that the identity of the explicit route need not carried with the packet.

Some routers analyze a packet's network layer header not to choose the packet's next hop, but also to determine a packet's precedence or class of service. MPLS allows but does not require the precedence or class of service to be fully or partially inferred from the label. In this case, one may say that the label represents the combination of a FEC and a precedence or class of service.

MPLS stands for Multiprotocol Label Switching, Multiprotocol because its techniques are applicable to any network layer protocol. In this thesis, however we focus on the use of IP as the network layer protocol. A router which supports MPLS is know as Label Switching Router, or LSR and a specific path through an MPLS network is a Label Switched Path, or LSP.

2.2 MPLS components

In this section, we introduce some of the basic concepts of MPLS and describe the general approach to be used.

The Internet Draft "Multiprotocol Label Switching Architecture" [30] defines a label as a short, fixed length, locally significant identifier which is used to identify a FEC. A label which is put on a particular packet represents the Forwarding Equivalence Class to which that packet is assigned.

A LSP is a specific path through an MPLS network. A LSP is supplied using a Label Distribution Protocol (LDP) such as Resource Reservation Protocol - Traffic Engineering (RSVP-TE) or Constaint-based Routing (CR-LDP). Either of these protocols will establish a path through an MPLS network and will reserve the necessary resources to meet pre-defined service requirements for the data path.

LSPs must be contrasted with traffic trunks. A traffic trunk is an aggregation of traffic flows of the same class which are placed inside a LSP. It is important, however,

to emphasize that there is a fundamental distinction between a traffic trunk and the path, and indeed the LSP, through which it traverses. The path through which a trunk traverses can be changed. In this respect, traffic trunks are similar to virtual circuits in ATM and Frame Relay networks.

A Label Distribution Protocol is a major part of MPLS and is a specification which lets a label switched router distribute labels to its LDP peers. When a LSR assigns a label to a forwarding equivalence class (FEC) it needs to let its relevant peers know of this label and its meaning and a LDP is used for this purpose. Since a set of labels from the ingress LSP to the egress LSR in an MPLS domain defines a Label Switched Path and since labels are mappings of network layer routing to the data link layer switched paths, LDP helps in establishing a LSP by using a set of procedures to distribute the labels among the LSR peers.

2.3 Label Switching Features

MPLS in conjunction with other standard technologies, offers many features:

The Interior Gateway Protocol (IGP) such as OSPF or IS-IS, is used to defined communication and the binding/mapping between FEC and next hop address. MPLS learns routing information from IGP (i.e., OSPF, IS-IS).

MPLS, in combination with the Border Gateway Protocol (BGP) provides support for highly scalable IP networks. IGP is used within Autonomous Systems (ASs), while Exterior Gateway Protocol (EGPs such as BGP) are used to interconnect ASs. The scalability feature of IGP makes it the best over the EGP and IS-IS is more scalable than OSPF. That is, a single OSPF area can support 150 or more routers and a single IS-IS area can support 500 or more routers.

It is best to first understand the benefits and disadvantages of each protocol, then use the network requirements to choose the IGP which best suites the needs. MPLS brings many other benefits to IP based networks. Some of the key benefits are discussed below.

Quality of Service (QOS)

IP QOS refers to the performance of an IP packet flow through one or more networks. The aim is to deliver end-to-end QOS to user traffic. IP QOS is characterized by a

small set of metrics, including the meausures of the ability to

- Guarantee a fixed amount of bandwidth for specific applications (such as audio/video conference applications).
- Control delay and delay variation, throughput, and packet loss rate.
- Provide specific, guaranteed and quantifiable Service Level Agreements (SLAs).

MPLS supports the same QOS as IP. But since connectionless networks cannot provide hard quality of service, it provides relative class-of-service transport only, which is unacceptable for services like voice and video which require a network with high predictability. MPLS adds a connection-oriented, or circuit-like behavior to native or traditional IP, in essence making it connection-oriented which enables hard QOS to be delivered.

Constraint-based and Congestion-aware Routing

Constraint-based and Congestion-aware routing are terms used to describe networks that are aware of their current utilization, existing capacity and provisioned services at all times. Traditional IP routing protocols, including OSPF, IS-IS and BGP, are not congestion-aware, and have to be modified to enable such awareness.

MPLS will modify traditional IP routing protocols to became constraint-based: once connections have been configured either by dynamic signaling or by static provisioning, the Layer 2 and Layer 3 network becomes aware of the amount of bandwidth being consumed, as well as the parts of the network being used to route the connections. This information can then be propagated to the IP routers, creating a congestion-aware view of the network and its current topology. All future network requests can be directed to their destination by not only the shortest path first (as defined OSPF), but by a path that will guarantee the bandwidth requirements of the IP application or service. Since CBR considers more than the topology of the network when computing routes, it may find a longer but lightly loaded path, which is better than using a heavily loaded path that may be shorter. Network traffic is distributed more evenly.

Traffic Engineering (TE)

Traffic Engineering refers to the process of selecting the paths chosen by data traffic in order to balance the traffic load on the various links, routers, and switches in the network. Traffic engineering is most important in networks where multiple parallel or

alternate paths are available.

The goal of TE is to compute a path from one node to another, such that the path does not violate the constraints (e.g. bandwidth requirements) and is optimal with respect to some metric. Once the path is computed, TE is responsible for establishing and maintaining forwarding state along such a path.

MPLS is strategically significant for Traffic Engineering because it can potentially provide most of the functionality available from the overlay model, in an integrated manner, and at a lower cost than the currently competing alternatives.

In MPLS networks, the traffic engineering building block is a Label Switched Path which can be manipulated and managed by the network operators to direct the traffic. The route for a given LSP can be established in two ways, a hop-by-hop LSP, or an explicitly routed (ER-LSP). When setting up a hop-by-hop LSP, each LSR independently chooses the next hop for each FEC. This is the usual mode today in existing IP networks.

In an explicitly routed LSP, each LSP does not independently choose the next hop; rather, a single LSR, generally the LSP ingress or the LSP egress, specifies several (or all) of the LSRs in the LSP. If a single LSR specifies the entire LSP, the LSP is strictly explicitly routed. If a single LSR specifies only part of the LSP, the LSP is loosely explicitly routed.

Loose ER-LSPs allow some flexibility for routing and rerouting options, and minimizes configuration overhead. In addition, a loose segment can be adaptive by moving to a new route according to the changes incurred in the Layer 3 routing table. However, this kind of route change is not always desirable due to the stability and control requirements of the network operators. In this case, the loose segment provides a mechanism, such that an alternative route will only be tried when failure happens.

The sequence of LSRs followed by an explicitly routed LSP may be chosen by configuration, or may be selected dynamically by a single node (for example, the egress node may make use of the topological information learned from a link state database in order to compute the entire path for the tree ending at that egress node).

Explicit routing may be useful for a number of purposes, such as policy routing or traffic engineering. In MPLS, the explicit route needs to be specified at the time that labels are assigned, but the explicit route does not have to be specified with each IP packet. This makes MPLS explicit routing much more efficient than the alternative of

IP source routing.

The attractiveness of MPLS for Traffic Engineering can be attributed to the following factors:

1. Explicit Label Switched paths which are not constrained by the destination based forwarding paradigm can be easily created through manual action or through automated action by the underlying protocols.
2. LSPs can potentially be efficiently maintained.
3. Traffic trunks can be instantiated and mapped onto LSPs. A set of attributes can be associated with traffic trunks which modulate their behavioural characteristics.
4. MPLS allows for both traffic aggregation and disaggregation whereas classical destination-only-based IP forwarding permits aggregation only.
5. It is relatively easy to integrate a constraint-based routing framework with MPLS.
6. A good implementation of MPLS can offer significantly lower overhead than competing alternatives for Traffic Engineering.

Additionally, through explicit label switched paths, MPLS permits a quasi- circuit switching capability to be imposed on the current Internet routing model.

Chapter 3

Optimal Path Discovery in MPLS Networks

Path discovery refers to the method used for selecting the LSP for a particular FEC in MPLS network. As mentioned in the previous chapter, the MPLS protocol architecture supports two options for route establishment: hop by hop routing and explicit routing.

This chapter explains the optimal path selection process which selects the best paths among the set of feasible paths discovered in the path computation.

3.1 Path Selection Information and Algorithms

This section reviews the basic building blocks of QOS path selection, namely the metrics on which the routing algorithm operates, and the path selection algorithm itself.

3.1.1 Metrics

The process of selecting a path that can satisfy the QOS requirements of a flow relies on both the knowledge of the flow's requirements and characteristics, and information about the availability of resources in the network. In addition, for purposes of efficiency, it is also important for the algorithm to account for the amount of resources the network has to allocate to support a new flow. In general, the network prefers to select the least cost path among all paths suitable for a new flow, and it may decide not to accept a new flow for which a feasible path exists, if the cost of the path is too high. Accounting

for these aspects involves several metrics on which the path selection process is based. They include:

- Link available bandwidth: We assume that most QOS requirements are derivable from a rate-related quantity termed *bandwidth*. We further assume that associated with each link is a maximal bandwidth value, namely the physical bandwidth or some fraction thereof that has been set aside for QOS flows. If a link is to accept a new flow with given bandwidth requirements, then at least that much bandwidth must be available on the link, and the relevant link metric is, therefore, the amount of available bandwidth.

- Link propagation delay: This quantity identifies high latency links which may be unsuitable for real-time requests. Link propagation delay can be used to eliminate specific links when selecting a path for a delay sensitive request.

- Hop-count: This quantity is a measure of the path cost to the network. A path with a smaller number of hops is typically preferable, since it consumes fewer network resources. As a result, the path selection algorithm will attempt to find the minimum hop path capable of satisfying the requirements of a given request.

3.2 Path selection

There are two major aspects to computing paths for QOS requests. The first is the path selection algorithm itself which determines the metrics and criteria that are used. The second aspect comes into play when the algorithm is invoked.

The optimization criteria used by the path selection are reflected in the costs associated with each interface in the topology of the network and how those costs are accounted for in the algorithm itself. The cost of a path is a function of both its hop count and the amount of available bandwidth. As a result, each interface has associated with it a metric, which corresponds to the amount of bandwidth that remains available on this interface. This metric is combined with hop count information to provide a cost value, which is used to select a path with the minimum number of hops that can support the request bandwidth. When several such paths are available, the preference is for the path whose available bandwidth (i.e., the smallest value on any of the links in the path) is maximal. The rationale for the above rule is the following: we focus on feasible paths that consume a minimal amount of network resources; and the rule for selecting

among these paths is meant to balance load as well as maximize the possibility that the required bandwidth is indeed available.

3.2.1 Path Computation Algorithm

Many practical path selection algorithms, are based on the notion of a *shortest path* between two nodes. Here each communication link is assigned a positive number called its *length*. A link can have a different length in each direction. A path (a sequence of links) between two nodes has a length equal to the sum of the lengths of its links. A shortest path routing algorithm routes each packet along a minimum length path between the origin and destination nodes of the packet. The simplest possibility is for each link to have unit length, in which case a shortest path is simply a path with minimum number of links (also called a min-hop path). More generally, the length of a link may depend on its bandwidth and its projected traffic load. The idea here is that a shortest path should contain relatively few and uncongested links, and therefore be desirable for routing.

A more sophisticated alternative is to allow the length of each link to change over time and to depend on the current congestion level of the link. Then a shortest path may adapt to temporary overloads and route packets around points of congestion. This idea is simple but contains some drawbacks, because by making link lengths dependent on congestion, we introduce a feedback effect between the routing algorithm and the traffic pattern within the network.

We implemented three standard algorithms for the shortest path problem: the Bellman algorithm, the Dijkstra algorithm, and the Floyd-Warshall algorithm. The first two algorithms find shortest paths from all nodes to a given destination node, and the third algorithm finds the shortest paths from all nodes to all other nodes.

An important distributed algorithm for calculating shortest paths to a given destination, known as the Bellman method has the form

$$D_i := \min_j (d_{ij} + D_j) \tag{1}$$

where D_i is the estimated shortest distance of node i to the destination and d_{ij} is the length of the link (i, j). Each node i executes this iteration with the minimum taken over all of its neighbours j. Thus $d_{ij} + D_j$ is the shortest distance from node i to the destination subject to the constraint of going through j, and $\min_j (d_{ij} + D_j)$ is the shortest distance from i to the destination going through the best neighbour. We will

proceed in the next section with a more detail description of the distributed Bellman's algorithm and the data structure used to record routing information. In practice, the Bellman iteration (3.1) can be implemented as an iterative process, that is, as a sequence of communications of the current value of D_j of nodes j to all their neighbors, followed by execution of the shortest distance estimate updates $D_i := \min_j(d_{ij} + D_j)$. A remarkable fact is that this process is very flexible with respect to the choice of initial estimates D_j and the ordering of communications and updates; it works correctly, finding the shortest distances in a finite number of steps, for an essentially arbitrary choice of initial conditions and for an arbitrary order of communications and updates. This allows an asynchronous, real-time distributed implementation of the Bellman method, which can tolerate changes of the link lengths as the algorithm executes.

3.2.2 Distributed Bellman Algorithm

Consider a routing algorithm that routes each packet along a shortest path from the packet's origin to its destination, and suppose that the link length may change either due to link failures and repairs, or due to changing traffic conditions in the network. It is therefore necessary to update shortest paths in response to these changes.

The idea is to compute the shortest distances from every node to every destination by means of a distributed version of the Bellman algorithm. An interesting aspect of this algorithm is that it requires very little information to be stored at the network nodes. Indeed, a node need not know the detailed network topology. It suffices for a node to know the length of its outgoing links and the identity of every destination.

We assume that each cycle has positive length. We also assume that the network always stays strongly connected, and that if (i, j) is a link, then (j, i) is also a link. We predict a pratical situation where the lengths d_{ij} can change with time. In the analysis, however, it is assumed that the lengts d_{ij} are fixed while the initial conditions for the algorithm are allowed to be essentially arbitrary. These assumptions provide an adequate model for a situation where the link lengths stay fixed after some time t_0 following a number of changes that occurred before t_0. We focus on the shortest distance D_i from each node i to a destination node taken for concreteness to be node 1. Under our assumptions, these distances are the unique solution of Bellman's equation,

$$D_i := \min_{j \in N(i)} (d_{ij} + D_j) \qquad i \neq 1 \tag{2}$$

$$D_1 = 0$$

where $N(i)$ denotes the set of current neighbors of node i.

The algorithm is well suited for distributed computation since the Bellman iteration (3.2) can be executed at each node i in parallel with every other node. The algorithm operates indefinitely by executing from time to time at each node $i \neq 1$ the iteration (3.2) using the last estimates D_j received from the neighbors $j \in N(i)$, and the latest status and lengths of the outgoing links from i. The algorithm also requires that each node i transmit from time to time its latest estimate D_i to all its neighbors. However, there is no need for either the iterations or the message transmissions to be synchronized at all nodes. Furthermore, no assumptions are made on the initial values $Dj, j \in N(i)$ available at each node i. The only requirement is that a node i will eventually execute the Bellman iteration (3.2) and will eventually transmit the result to the neighbours. Thus, a totally asynchronous mode of operation is envisioned.

It turns out that the algorithm is still valid when executed asynchronously as described above. If a number of link length changes occur up to some time t_0, and no other changes occur subsequently, then within finite time from t_0, the asynchronous algorithm finds the correct shortest distance of every node i. The shortest distance estimates available at time t_0 can be arbitrary numbers, so it is not necessary to reinitialize the algorithm after each link status or link length change.

We now state formally the distributed, asynchronous Bellman algorithm and proceed to establish its validity. At each time t, a node $i \in 1$ has available:

$D_j^i(t)$ = Estimate of the shortest distance of each neighbor node $j \in N(i)$ which was last communicated to node i

$D_i(t)$ = Estimate of the shortest distance of $i$which was last computed at node i according to the Bellman iteration

The distance estimates for the destination node 1 are defined to be zero, so

$$D_1(t) = 0, \qquad \text{for all } t \geq t_0$$

$$D_1^i = 0, \qquad \text{for all } t \geq t_0, \text{ and } i \text{ with } 1 \in N(i)$$

Each node i also has available the link lengths d_{ij}, for all $j \in N(i)$, which are assumed constant after the initial time t_0. We assume that the distance estimates do not change

except at some times $t_0, t_1, t_2, \ldots$, with $t_{m+1} > t_m$, for all m, and $t_m \to \infty$ as $m \to \infty$ when at each processor $i \neq 1$, one of three events happens:

1. Node i updates $D_i(t)$ according to

$$D_i(t) := \min_{j \in N(i)} (d_{ij} + D_j^i(t))$$

   and leaves the estimates $D_i^i(t), j \in N(i)$, unchanged.

2. Node i receives from one or more neighbors $j \in N(i)$ the value of D_j which was computed at node j at some earlier time, updates the estimate D_j^i, and leaves all other estimates unchanged.

3. Node i is idle, in which case all estimates available at i are left unchanged.

Chapter 4

The Flow Deviation Algorithm: Finding optimal link flows in IP networks

In this chapter we discuss the Flow Deviation Algorithm (FDA) designed to minimize the network average time delay. The emphasis of the algorithm is on two aspects of the routing problem. The first has to do with selecting routes to achieve optimal performance. The second aspect of the problem is how the flow requirements are distributed among the links of the network in order to minimize the network delay.

4.1 General approach

Consider a physical network consisting of a set of N nodes denoted by $\mathcal{N}$ and a set of L physical links denoted by $\mathcal{L}$. The traffic requirements are specified by an $N \times N$ matrix $R = r_{ij}$, called the requirement matrix, whose entries are non-negative. Let $C_{i,j}$ denote the capacity in bandwidth units of the physical link from an origin node i to a destination node j. The set of routes connecting O-D pair (o, d) is denoted by $\mathcal{R}_{o,d}$. Each route consists of a non-cycling sequence of physical links.

Messages are offered to O-D pair (i, j) according to a Poisson process with mean rate λ_{ij}. The average message length from node i to node j is exponentially distributed with mean $1/\mu_{ij}$. Let $\rho_{ij} = \lambda_{ij}/\mu_{ij}$ denote the intensity of the offered traffic stream.

Let f_r denote the flow on route r. The total flow F_{ij} on link (i,j) is denoted by

$$F_{ij} = \sum_{r \in \mathcal{A}_{ij}} f_r$$

where $\mathcal{A}_{ij}$ is the set of routes that use link (i,j).

We are interested in the numerical solution of the following network flow problem:

Minimize: The average end to end network delay.

$$T = \sum_{(i,j)} \frac{F_{ij}}{\gamma} T_{ij} \tag{1}$$

where $\gamma = \sum_{(i,j)} \lambda_{ij}$ is the total message arrival rate from external sources (bits/sec) and T_{ij} is the average delay experienced by a message on link (i,j) (sec)

subject to:

$$0 \leq F_{ij} \leq C_{ij} \qquad \forall i,j \in \mathcal{N} \tag{2}$$

With reference to equation (4.1) T_{ij} is the sum of two components:

$$T_{ij} = T_{ij}^1 + T_{ij}^2$$

where, assuming that the link is modelled as an $M/M/1$ queue,

$$T_{ij}^1 = \frac{1}{\mu_{ij} C_{ij} - F_{ij}}$$

is the transmission and queue delay, and $T_{ij}^2 = p_{ij}$ is the propagation delay. If the propagation delay is negligible, then the average network delay from Eq.(4.1) is:

$$T = \sum_{(i,j)} \frac{F_{ij}}{\gamma} \left(\frac{1}{\mu_{ij} C_{ij} - F_{ij}} \right) \tag{3}$$

If the propagation delay is not negligible, then :

$$T = \frac{1}{\gamma} \sum_{(i,j)} \left(\frac{F_{ij}/\mu_{ij}}{\mathrm{C}_{ij} - F_{ij}/\mu_{ij}} + (F_{ij}/\mu_{ij}) \mu_{ij} p_{ij} \right) \tag{4}$$

We choose to write this expression in terms of average data rate by defining the flow on link (i,j) to be $F_{ij} := F_{ij}/\mu_{ij}$. Equation (4.4) becomes:

$$T = \frac{1}{\gamma}\sum_{(i,j)}\left(\frac{F_{ij}}{C_{ij}-F_{ij}} + F_{ij}p_{ij}^1\right) \tag{5}$$

where

$$p_{ij}^1 = \mu_{ij}p_{ij}$$

An expression of the form

$$\sum_{(i,j)} D_{ij}(F_{ij}) \tag{6}$$

where each function D_{ij} is monotonically increasing, is often appropriate as a cost function for optimization. This assumes that one achieves reasonably good routing by optimizing the average levels of link traffic without paying attention to other aspects of the traffic statistics. Thus, the cost function $\sum_{(i,j)} D_{ij}(F_{ij})$ is insensitive to undesirable behavior associated with high variance and with correlations of packet interarrival times and transmission times. A frequently used formula is

$$D_{ij}(F_{ij}) = \frac{F_{ij}}{\mathrm{C}_{ij}-F_{ij}} + F_{ij}p_{ij}^1. \tag{7}$$

where p_{ij}^1 is the propagation delay.

Another cost function with similar qualitative properties is given by

$$\max_{(i,j)}\left\{\frac{F_{ij}}{C_{ij}}\right\} \tag{8}$$

(maximum link utilization). A study [13] has shown that it typically makes little difference whether the cost function of Eq.(4.7) or that of Eq.(4.8) is used for routing optimization. This indicates that one should employ the cost function that is easiest to optimize. In what follows we concentrate on cost functions of the form $\sum_{(i,j)} D_{ij}(F_{ij})$.

4.2 Characterization of optimal routing

We now formulate the problem of optimal routing. The main objective in this section is to show that optimal routing directs traffic along paths which are shortest with respect to link lengths.

Recall the form of the cost function

$$\sum_{(i,j)} D_{ij}(F_{ij}) \tag{9}$$

where F_{ij} is the total flow (in bits per second) carried by link (i,j) and is given by

$$F_{ij} = \sum_{p \in \mathcal{A}_{ij}} x_p \tag{10}$$

where x_p is the flow (in bits per second) of path p and $\mathcal{A}_{ij}$ is the set of paths that use link (i,j). For each pair $w = (i,j)$ of distinct nodes i and j, there are the constraints

$$\sum_{p \in P_w} x_p = r_w \tag{11}$$

$$x_p \geq 0, \qquad \text{for all } p \in P_w \tag{12}$$

where r_w is the traffic offered to the OD pair w and P_w is the set of paths which connect the OD pair w. W is the set of all OD pairs. In terms of the unknown path flow vector, $\mathbf{x} = \{x_p \mid p \in P_w, w \in W\}$ the optimization problem is written as

Minimize

$$\sum_{(i,j)} D_{ij}\left(\sum_{p \in \mathcal{A}_{ij}} x_p\right)$$

subject to

$$\sum_{p \in P_w} x_p = r_w \qquad \text{for all } w \in W \tag{13}$$

$$x_p \geq 0, \qquad \text{for all } p \in P_w$$

In what follows we will characterize the optimal routing in terms of the first derivative D'_{ij} with respect to F_{ij} of the function D_{ij}. We assume that each D_{ij} is a differentiable function of F_{ij} and is defined on an interval $[0,C_{ij})$, where C_{ij} is either a positive number (typically representing the link capacity) or else ∞; D_{ij} is convex, continuous, and has strictly positive and continuous first and second derivatives on $[0,C_{ij})$, where the derivatives at 0 are defined by taking the limit from the right. Furthermore, $D_{ij}(F_{ij}) \to \infty$ as $D_{ij}(F_{ij}) \to C_{ij}$.

Let $\mathbf{x}$ be the vector of path flows x_p. Denote by $D(\mathbf{x})$ the cost function of the problem Eq.(4.13),

$$D(\mathbf{x}) = \sum_{(i,j)} D_{ij}\Big(\sum_{p \in \mathcal{A}_{ij}} x_p\Big)$$

The object of the routing optimization is to find each $x_p \geq 0$ satisfying the conservation equations, such that $D(\mathbf{x})$ is minimized. One additional constraint is the capacity constraint $F_{ij} \leq C_{ij}$. This constraint serves as a penalty function on the time delay to be minimized and is automatically brought into play when F_{ij} approaches C_{ij}. A configuration $\mathbf{x}$ is *feasible* if it satisfies the constraints (2) and (13).

Let $\partial D(\mathbf{x})/\partial x_p$ denote the partial derivative of D with respect to x_p. Then

$$\frac{\partial D(\mathbf{x})}{\partial x_p} = \sum_{(i,j) \in p} D'_{ij}$$

where the first derivatives D'_{ij} are evaluated at the total flows corresponding to $\mathbf{x}$. We regard $\partial D(\mathbf{x})/\partial x_p$ as the length of the path p when the length of each link (i,j) is taken to be the first derivative D'_{ij} evaluated at $\mathbf{x}$. In what follows $\partial D(\mathbf{x})/\partial x_p$ is called the *first derivative length of path p.*

Let $\bar{\mathbf{x}} = \{\bar{x_p}\}$ be an optimal path flow vector. Then if $\bar{x_p} > 0$ for some path p of an OD pair, shifting a small amount of flow $\delta > 0$ from path p to any other path p' of the same OD pair will increase the cost; otherwise the optimality of $\bar{\mathbf{x}}$ would be violated. The change in cost from this shift is

$$\delta\Big(\frac{\partial D(\bar{\mathbf{x}})}{\partial x_{p'}} - \frac{\partial D(\bar{\mathbf{x}})}{\partial x_p}\Big)$$

and since this change must be nonnegative, we obtain

$$\bar{x}_p > 0 \qquad \Rightarrow \qquad \frac{\partial D(\bar{\mathbf{x}})}{\partial x_{p'}} \geq \frac{\partial D(\bar{\mathbf{x}})}{\partial x_p}, \qquad \text{for all } p' \in P_w \tag{14}$$

The condition (4.14) is a necessary condition for optimality of $\bar{\mathbf{x}}$. It can also be shown to be sufficient for optimality if the functions D_{ij} are convex.

The implementation of the Flow Deviation Algorithm starts with zero flow on each link and assigns lengths to the links based on their first derivative of delay with respect to the flow in the links. Each flow assignment is performed in one step of the FDA. In

each step, Bellman's algorithm [3] is used to find the shortest routes between all OD pairs.

The requirements are assigned to the shortest routes between all OD nodes to yield an initial global flow, Gflow which is not optimal and in many cases, it is not even feasible in the sense that it does not satisfy the link capacity constraints. In this case, the link capacities are increased so that the link flow satisfies the constraints. In each new iteration, capacities are calculated to ensure that the global flows are feasible. In each step, we assign link lengths based on the first derivative of delay with respect to flow, calculate the shortest paths which may be different from paths calculated earlier and assign the flows to these paths to yield Eflow, the current extremal flow. Gflow is the previous calculated best flow, and Eflow is the extremal flow calculated in the current step. We now add a small amount of the new Eflow to improve the old Gflow. It is obvious that every linear combination of two valid flows is a valid flow. If we have flow vectors $\mathbf{x}$ and $\bar{\mathbf{x}}$ (representing respectively Gflow and Eflow), and some number $\alpha \in [0,1]$, then $\alpha\ (\bar{\mathbf{x}}) + (1-\alpha)\mathbf{x}$ will also be a valid flow, and for some value of α, the new flow might be better in terms of delay than the original flow. The value of α minimizing the average total network delay is found by performing a line search :

$$D[\alpha^*(\bar{\mathbf{x}}) + (1-\alpha^*)\mathbf{x}] = \min_{\alpha\in[0,1]} D[\alpha(\bar{\mathbf{x}}) + (1-\alpha)\mathbf{x}] \tag{15}$$

where α^* is the stepsize that minimizes the function $\mathrm{D}[\alpha^*(\bar{\mathbf{x}}) + (1-\alpha^*)\mathbf{x}]$

The new set of path flows is obtained by

$$x_p := \alpha(\bar{x_p}) + (1-\alpha)x_p \qquad \text{for all } p \in P_w, w \in W \tag{16}$$

and the process is repeated. The process, in the most general case, goes through two stages:

1. The capacities are not sufficient to handle the flow and therefore must be adjusted.
2. The flow is redistributed in such a way that the capacities are sufficient.

The delay calculated in stage 1 is not a real delay because the capacities were adjusted. The delay calculated in stage 2 is real and will decrease at each step of the algorithm. The algorithm will stop in stage 1 only if it fails to find a feasible solution. In stage 2, the algorithm will end when the delay stops decreasing.

A description of the Flow Deviation Algorithm is as follow:

Step 1: Initialize the link lengths.

Step 2: Find the initial set of shortest routes based on these lengths. For each OD pair, use Bellman's algorithm [3] to find the shortest routes.

Step 3: Assign the initial global flows to the links of the shortest routes.

Step 4: Adjust the link capacities to ensure that the global flows are feasible.

Step 5: Update the link lengths.

Step 6: Find the set of shortest routes.

Step 7: Assign the extremal flow to the links of the shortest routes.

Step 8: Find the value of α in the range $0 \leq \alpha \leq 1$ such that the flow $\alpha(\bar{x_p}) + (1-\alpha)x_p$ minimizes the total network delay.

Step 9: Compute a new global flow. The new flow is an improvement on the previous flow when applied to the same link capacities.

Step 10: Adjust the link capacities to ensure that the global flows are feasible.

Step 11: Calculate the average total network delay.

Step 12: Stopping rule. If the delay stops decreasing then halt, else go to step 5.

Consider the small network shown in figure 4.1. There are three nodes and six directed links. All links have capacities equal to 2. There are two requirements, a (0,1) requirement of magnitude 2 and a (1,2) requirement of magnitude 3. The objective is to find minimum delay link flows using the FDA. Table 3.1 shows the progress of the FDA through 46 iterations. Each column shows the optimal link flow and the average total delay. The FDA computes a minimum value for the cost function although its convergence rate near the optimum tends to be very slow.

4.3 An Efficient Flow Deviation Algorithm

The optimal routing problem (4.13) can be converted to a problem involving only positivity constraints by expressing the flows of the minimum first derivative length (MFDL) paths in terms of other flows, while eliminating the equality constraints

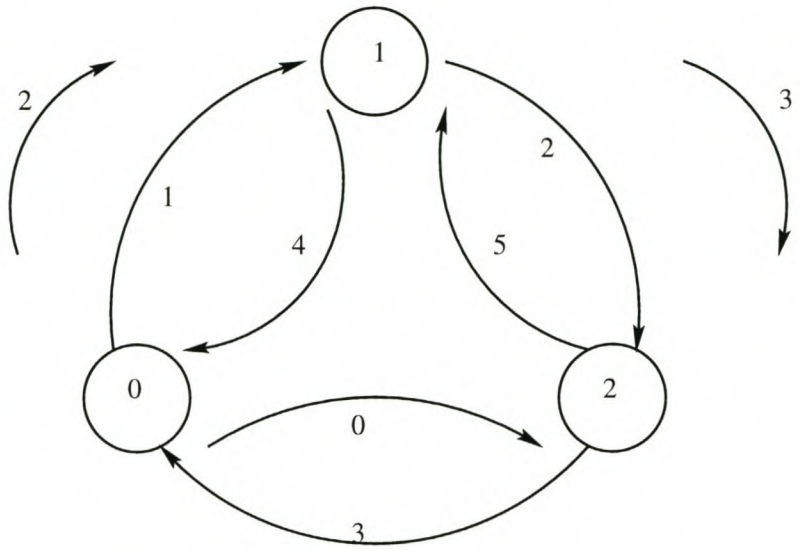


Figure 4.1: Minimum delay routing problem

Links	It.3	It.5	It.15	It.25	It.35	It. 46
0	1.442	1.465	1.556	1.609	1.632	1.646
1	1.788	1.778	1.722	1.689	1.671	1.657
2	1.230	1.243	1.279	1.296	1.303	1.304
3	1.769	1.756	1.72	1.703	1.696	1.695
4	0	0	0	0	0	0
5	0.558	0.534	0.443	0.393	0.367	0.353
delay	4.070	3.928	3.568	3.46	3.439	3.432

Table 4.1: Link flows in the FDA

$$\sum_{p \in P_w} x_p = r_w$$

in the process. For each OD pair w, let $\bar{p}_w$ be the MFDL path with the respect to the current length $D'_{ij}(F_{ij})$. For each w, $x_{\bar{p}_w}$ (flow of the MFDL path $\bar{p}_w$) is substituted in the cost function $D(\mathbf{x})$ using the equation

$$x_{\bar{p}_w} = r_w - \sum_{\substack{p \in P_w \\ p \neq \bar{p}_w}} x_p \tag{17}$$

thereby obtaining a problem of the form

Minimize $\tilde{D}(\tilde{\mathbf{x}})$ subject to

$$x_p \geq 0, \qquad \text{for all } w \in W, p \in P_w, p \neq \bar{p}_w \tag{18}$$

where $\tilde{\mathbf{x}}$ is the vector of all paths flows which are not MFDL paths.

We now calculate the derivatives that will be needed to the problem of Eq.(4.18). Using Eq.(4.17) and the definition of $\tilde{D}(\tilde{\mathbf{x}})$, we obtain

$$\frac{\partial \tilde{D}(\tilde{\mathbf{x}})}{\partial x_p} = \frac{\partial D(\mathbf{x})}{\partial x_p} - \frac{\partial D(\mathbf{x})}{\partial x_{\bar{p}_w}}, \qquad \text{for all } p \in P_w, p \neq \bar{p}_w \tag{19}$$

for all $w \in W$. In Section 4.2 we saw that $\partial D(\mathbf{x})/\partial x_p$ is the first derivative length of path p, that is,

$$\frac{\partial D(\mathbf{x})}{\partial x_p} = \sum_{(i,j)\in p} D'_{ij} F_{ij}, \tag{20}$$

Regarding second derivatives, differentiation of the first derivative expressions (4.19) and (4.20) shows that

$$\frac{\partial^2 \tilde{D}(\tilde{\mathbf{x}})}{(\partial x_p)^2} = \sum_{(i,j)\in L_p} D''_{ij}(F_{ij}) \qquad \text{for all } w \in W, p \in P_w \tag{21}$$

where, L_p is the set of links belonging to either p, or $\bar{p}_w$, but not both. The length of each link is defined as the first derivative of the total delay with respect to the flow in link and the path lengths are simply the sum of the link lengths in that path.

L_p is formed in this way because each link has a second derivative length and there are some links that belong to either the nonshortest path p or to the shortest path $\bar{p_w}$, but not to both. No path uses the same link more than once.

Expressions for both the first and second derivatives of the reduced cost $\tilde{D}(\tilde{\mathbf{x}})$, are now available and thus the scaled projection method can be applied. The iteration takes the form

$$x_p = x_p - \alpha H_p^{-1}(d_p - d_{\bar{p}_w}), \qquad \text{for all } w \in W, p \in P_w, p \neq \bar{p}_w \tag{22}$$

where d_p and $d_{\bar{p}_w}$ are the first derivative length of the paths p and $\bar{p_w}$ given by Eq.(4.23)

$$d_p = \sum_{(i,j)\in p} D'_{ij}(F_{ij}), \qquad d_{\bar{p}_w} = \sum_{(i,j)\in \bar{p}_w} D'_{ij}(F_{ij}) \tag{23}$$

and H_p is the second derivative path length

$$H_p = \sum_{(i,j)\in L_p} D''_{ij}(F_{ij}) \tag{24}$$

given by Eq.(4.24). The parameter α is a positive scalar which may be chosen by a variety of methods. In the original statement of the Bertsekas-Gallager algorithm in [3], the authors suggest setting α to 1 and then decreasing it by a constant factor as the algorithm proceeds. Because many requirements are moved before recomputing link lengths in the implementation, see Kershenbaum [1], the algorithm works better if α is set to a smaller value.

Like the FDA algorithm, our implementation starts with link lengths calculated with zero flow and finds the shortest paths between all pairs of nodes. These paths form the initial set of paths. We next set the flow on each path equal to the requirement between its ends and we add this requirement to the flows on the links of the path. As a result the flow on a link becomes a sum of flows on all paths which contain this link.

In our implementation of the EFDA algorithm, each path is stored as a vector of its links. We keep the vectors sorted in ascending order as in Bertsekas-Gallager [1], which allows us to compare two paths by comparing vectors element by element. However, the original implementation presented in [1] used linked lists instead of vectors. No path can be longer than NL (total number of links), and we allocate NL elements for each path. In fact, N (number of nodes) can be used because nodes do not occur twice in paths.

An important difference between our implementation and the Bertsekas-Gallager version [1] is that by using a route generation procedure as part of the shortest path algorithm, routes with least cost can be recorded as they are generated.

The use of second derivatives improves the rate of convergence and facilitates stepsize selection in the optimization algorithm. This procedure is to scale the descent direction using second derivatives of the objective function as in the Bertsekas-Gallager Algorithm.

The algorithm executes a sequence of iterations. On each iteration, for each pair OD between which there is a non-zero requirement, we calculate the link lengths based on

the current flows and find the shortest path. Then we move some flow from all the paths onto this shortest path, and the amount being moved is calculated in Eq.(4.22).

The algorithm is described schematically as follow:

Step 1: Assign the link lengths based on the first derivative of delay with respect to flow starting with zero flows.

$$d_p = \sum_{(i,j)\in p} D'_{ij}(F_{ij})$$

Step 2: Find shortest paths using Bellman's algorithm [3] for each OD pair.

Step 3: Load the shortest path for every pair of requirements.

Perform the iterations:

Step 4: Adjust the link capacities if necessary to ensure that the path flows are feasible.

Step 5: Assign new link lengths based on first derivative of delay with respect to current flow. The new flow is an improvement on the previous flow when applied to the same link capacities.

Step 6: Find shortest paths for every OD pair.

Step 7: Add new path to the path set and compute how much flow must be moved to it. The amount of flow δ to move off of path p is computed as

$$\delta = \alpha(d_p - d_{\bar{p_w}})/H_p$$

Step 8: For each OD pair, move the flow from all other paths to the shortest paths.

Step 9: Calculate the new network average delay.

The iteration stops when the current delay is no longer significantly less than the previous delay. To prevent infinite iteration, the algorithm also stops when the new factor of capacity adjustment is not significantly less than previous one.

Again consider the 3 node network shown in Figure 4.1 and compare the results in Table 4.2 with those reported in previous section.

Table 4.2 shows the output of the EFDA algorithm through 38 iterations. The EFDA algorithm converges faster than the FDA algorithm and a slightly better result is obtained. We applied the EFDA algorithm to a network consisting of 6 nodes 30 links

Links	It.3	It.5	It.15	It.20	It.25	It.30	It.35	It.38
0	1.54522	1.3965	1.4131	1.4497	1.4984	1.5327	1.5574	1.6545
1	0.5534	0.7973	1.2110	1.3629	1.4860	1.5467	1.5853	1.6529
2	0.0986	0.1938	0.6241	0.8126	0.9845	1.0794	1.1427	1.3074
3	2.9013	2.8061	2.3758	2.1817	2.0154	1.9205	1.8572	1.6925
4	0	0	0	0	0	0	0	0
5	0.4547	0.6034	0.5868	0.5502	0.5015	0.4672	0.4425	0.3454
delay	4.2877	4.3388	4.6161	4.8673	5.3039	5.1570	3.9000	3.4306

Table 4.2: Link flows in EFDA

presented in [2]. Only 18 iterations are required to converge while the FDA requires 1075 iterations for convergence.

The reason for the rapid convergence is that EFDA works with one OD pair (one requirement) at a time, calculates path lengths and moves flow from one path to another. The FDA algorithm moves flow from all requirements at the same time. Another important feature is that we compute an approximation to the second derivative of delay with respect to flow and use this as a correction factor on the amount of flow to move instead of performing a line search. This allows us to efficiently compute a reasonable estimate of the amount of flow to move.

The following observations can be made regarding the EFDA algorithm:

1. Since $d_p \geq d_{p_w}$ for all $p \neq \bar{p}_w$, all the nonshortest path flows that are positive will be reduced with the corresponding increment of flow being shifted to the MFDL path $\bar{p}_w$. If α is large enough, all flow from nonshortest paths will be shifted to the shortest path. The delay then increases and will falsely indicate that the algorithm has converged. Therefore the algorithm may be viewed as a generalization of the adaptive routing method based on shortest paths with α, H_p, and $d_p - d_{p_w}$ determining the amount of flow shifted to the shortest path. With α small, the algorithm tends not to oscillate as much, that is, moving flow back and forth among the same links. If α is too small, however, the convergence slows as the algorithm moves flow from one path to another in many small steps instead of fewer larger ones. At high utilizations, it becomes important to prevent oscillation.

2. Those nonshortest path flows x_p, $p \neq \bar{p}_w$ that are zero will stay zero. Therefore,

the path flow iteration of Eq.(4.22) should only be carried out for paths that carry positive flow.

3. Only paths that carried positive flow at the starting flow pattern or were MFDL paths at some previous iteration can carry positive flow at the beginning of an iteration. This is important since it tends to keep the number of flow-carrying paths small, with a corresponding reduction in the amount of calculation and bookkeeping needed at each iteration.

4.4 Multiservice network dimensioning

Up to this point, we have considered *single-service* networks, that is, networks for which a call occupies one circuit in each link along its routes. The focus of this section is on *multiservice networks.*

Multiservice networks carry calls which belong to several call classes with different bandwidth requirements – a telephone call for example requires one unit of transmission capacity whereas a video call may require hundreds of units of capacity. In this section, the EFDA algorithm is extended to investigate the performance of optimal routing in multiservice networks carrying several classes of traffic each with different bandwidths and different quality of service requirements.

4.4.1 Analytic Techniques

We consider the same network in section 4.1 which consists of N nodes with L physical links. Recall that the design problem is to find optimal flows that would optimize the objective function.

$$T = \sum_{(i,j)} \frac{F_{ij}}{\gamma} T_{ij} \tag{25}$$

where F_{ij} is the flow on the link (i,j) in message/sec and T_{ij} is the average delay experienced by a message on link (i,j). The original Flow Deviation Algorithm used an objective function based on the $M/M/1$ queue. This queue assumes that the packets arrive according to a Poisson process and that the packet lengths are exponentially distributed. In the single service network the total delay on the link, (i,j) with service time T_s and utilization U_{ij} is

$$T_{ij} = \frac{T_s}{1 - U_{ij}}, \tag{26}$$

where T_s is the average length of a message of size M, divided by the capacity of the link C_{ij}, and U_{ij} is the flow in the link, F_{ij} divided by C_{ij}. Thus,

$$T_{ij} = \frac{M/C_{ij}}{1 - F_{ij}/C_{ij}} = \frac{M}{C_{ij} - F_{ij}} \tag{27}$$

The weighted network delay is therefore

$$T = \sum_{(i,j)} \frac{MF_{ij}}{C_{ij} - F_{ij}} \tag{28}$$

where M is the message length. In a multiservice network, the inputs of the models correspond to those of single class models. The additional consideration is the specification of the link service discipline which is the rule for selecting the next customer to receive service.

Each link in our multiservice problem will be modelled as a processor sharing queue in which the total service capacity is equally shared between the available customers.

For the Process Sharing the total average system response time for a class-k, where $k \in K$, is :

$$T = \sum_{(i,j)} \sum_{k} \frac{M_k F_{ijk}}{C_{ij} - \sum_{k} M_k F_{ijk}} \tag{29}$$

where M_k is the length of a class-k message in the system, and F_{ijk} is the class k flow on link (i, j).

4.4.2 Service integration

The multiservice traffic is formed as follows: The class k requirement λ_{ij}^k between two given nodes (i, j) is equal to the base traffic intensity λ_{ij} multiplied by a class-dependent traffic intensity factor γ_k multiplied by the bandwidth requirement b_k for this service.

Links	class 1	class 2
0-1	0.248	1.492
0-2	0.249	1.494
1-0	0.197	1.186
1-2	0.252	1.513
2-0	0	0
2-1	0	0

Table 4.3: Distribution of flows per service class

Consider the small network shown in Figure 4.1. The network consists of three nodes and six directed links. All links have capacities equal to 2. The objective is to find minimum delay routes using the EFDA in a multiservice network. There is a basic traffic intensity of 2.0 between node 0 and 1 and 3.0 between node 1 and 2.

There are two classes of service, with bandwidths b_1=1 and b_2=3. The message length for class one service is M_1=1 and M_2=2 for the second class. The requirement for class 1 from node 0 to node 1 is therefore equal to 0.5 and from node 1 to node 2 equals to 0.75. For class 2, the requirement from the node 0 to node 1, is equal to 1.5 and from node 1 to node 2, the requirement is equal to 2.25. We assume that the class-dependant traffic intensity factor γ to be 0.25 for both classes. For the overall network delay, the experiment gave 4.48 seconds as minimal delay and the distribution of flows per class are given in Table 4.3.

4.4.3 A Larger Network

We also investigated the optimal flows for a larger network consisting of 8 nodes which is a fictitious representation of the NSF ATM backbone network introduced by Mitra [2]. The topology of the network is shown in Fig.5.1. Each link carries traffic in one direction. The transmission capacity of each uni-directional link is 2812 bandwidth units. The double lines indicate two-unidirectional links each having a transmission capacity of 5624 bandwidth units.

The network carries six traffic classes: the bandwidth requirement of the first service is 1 unit and the bandwidth of services 2 through 6 are 3, 4, 6, 24, and 40 respectively. Table 4.5 presents the different message lengths per service while the base traffic intensity matrix is shown in table 4.4. The class dependant traffic intensity is $\rho_{ij}^s = \rho_{ij}\gamma_s b_s$. Figures 4.2 and Fig 4.3 present the optimal link flows per service class.

nodes	1	2	3	4	5	6	7	8
1	–	6	7	1	9	5	2	3
2	7	–	24	3	31	15	6	9
3	8	25	–	4	37	18	7	11
4	1	3	3	–	4	7	1	1
5	11	33	39	5	–	24	9	15
6	5	14	16	2	21	–	4	6
7	2	5	6	1	8	4	–	2
8	3	8	10	1	12	6	2	–

Table 4.4: Traffic intensity matrix

	class 1	class 2	class 3	class 4	class 5	class 6
γ_s	0.4	0.4	1.0	0.5	0.5	0.1
b_s	1	3	4	6	24	40
M_s	1	3	2	3	1	1

Table 4.5: Class-dependent factor and slots per service

When one considers the utilisation of the links 2-3, 4-5 and 5-7, one can see that it is much higher compared with the utilisation of the other links. However their flows are not large compared with the other links (see Fig 4.2 and Fig 4.3).

Although links 3-4 and 7-8 have large flows, the utilisations are moderate. The reason is that they have large capacities, 5624 bandwidth units as opposed to 2812.

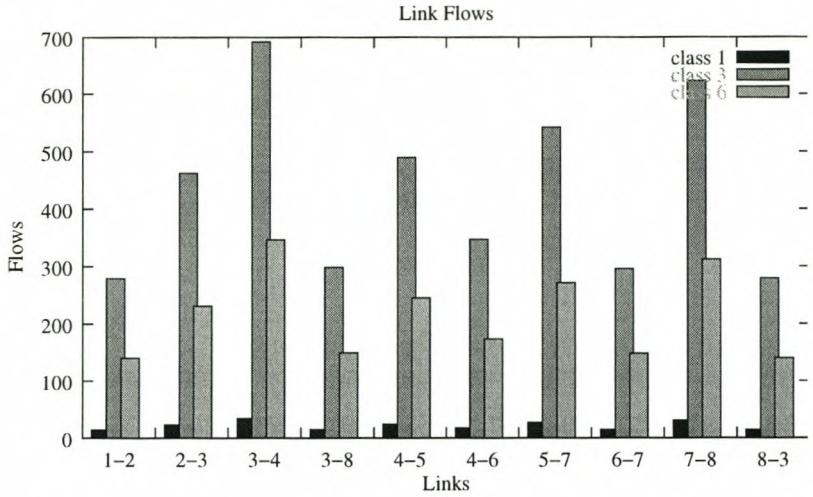


Figure 4.2: The optimal link flows per class

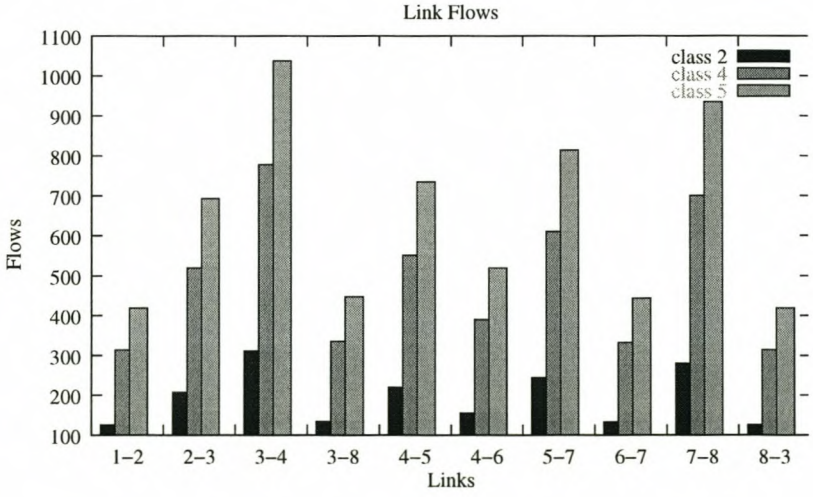


Figure 4.3: The optimal link flows per class

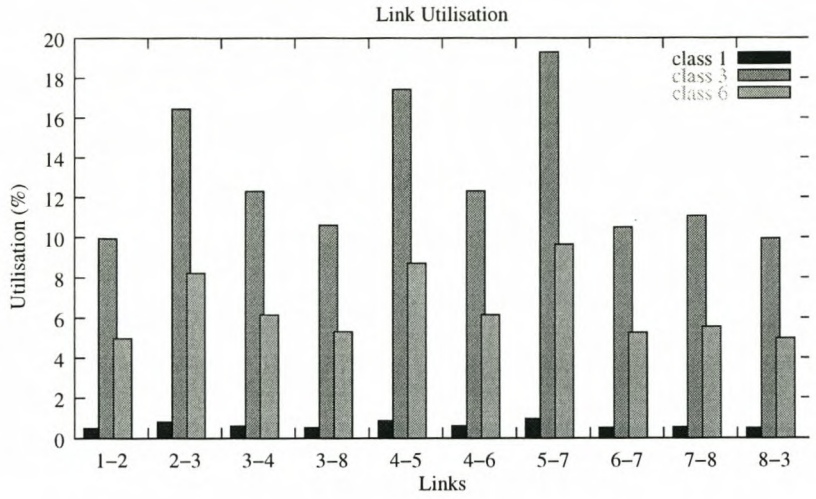


Figure 4.4: Link utilisation per class

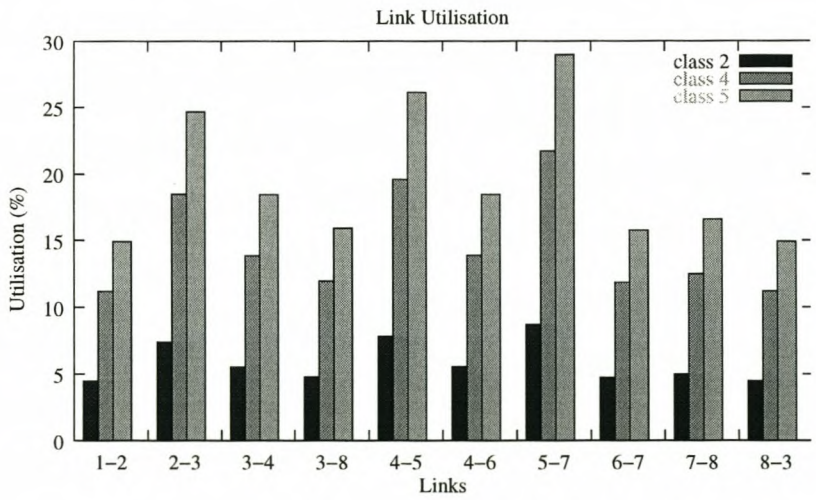


Figure 4.5: Link utilisation per class

Chapter 5

Optimal Link Capacities in IP Networks

The general topology optimization problem is complex and concerns the optimum selection of links, the assignment of capacities to these links and the routing of requirements on these links. The routing problem was discussed in the previous chapter. In this chapter, the focus is on the optimal assignment of capacities to the links and the routing of requirements on these links.

5.1 Capacity assignment problem

This section begins by considering the problem of assigning optimal capacities to the links in the network given the link topology and link flows.

Consider a network consisting of N nodes and L links. Let $i-j$ denote the link connecting OD pair (i,j). Link $i-j$ has capacity C_{ij} measured in bits/sec.

There are S classes of messages. The traffic requirements between the node pairs are measured in bits per second. We assume a flow distribution – a flow on each link and for each class which satisfies the requirements.

The objective is to compute the optimal link capacities for a network where the topology and traffic flows are known and fixed which minimize the average delay subject to the linear total cost of the system:

$$\sum_{(i,j)} d_{ij}C_{ij} \tag{1}$$

where d_{ij} is the positive cost per unit capacity on link (i, j).

To minimize the objective function, we proceed by using a Lagrange multiplier β and by forming the Lagrangian function as follows:

$$L = T + \beta\Big(\sum_{(i,j)} d_{ij}C_{ij} - D\Big) \tag{2}$$

where D is the total cost of the network and T is given by the $M/M/1$ delay function:

$$T = \frac{1}{\gamma}\sum_{(i,j)}\Big(\frac{F_{ij}}{C_{ij} - F_{ij}}\Big)$$

In Eq.(5.38), if we find the minimum value of L with respect to the capacity assignment, then we will have found the solution to the capacity assignment problem.

As is usual in Lagrangian problems, we set the partial derivatives $\partial L/\partial C_{ij}$ to zero:

$$\frac{\partial L}{\partial C_{ij}} = \beta d_{ij} - \frac{F_{ij}}{\gamma(C_{ij} - F_{ij})^2} = 0$$

Solving for C_{ij} gives:

$$C_{ij} = F_{ij} + \frac{1}{\sqrt{\beta\gamma}}\sqrt{\frac{F_{ij}}{d_{ij}}} \tag{3}$$

The objective now is to find the value of β. Once we have evaluated the constant β, this will be our solution.

$$\sum_{(i,j)} d_{ij}C_{ij} = \sum_{(i,j)}\Big(F_{ij}d_{ij} + \frac{1}{\sqrt{\beta\gamma}}\sqrt{F_{ij}d_{ij}}\Big)$$

From this equation, solving for β gives,

$$\frac{1}{\sqrt{\beta\gamma}} = \frac{D - \sum\limits_{(i,j)} F_{ij}d_{ij}}{\sum\limits_{(i,j)} \sqrt{F_{ij}d_{ij}}}$$

Using this last form in our Eq.(5.39), the optimal solution to the capacity assignment problem is

$$C_{ij} = F_{ij} + \frac{D - \sum_{(i,j)} F_{ij}d_{ij}}{\sum_{(i,j)} \sqrt{F_{ij}d_{ij}}} \sqrt{\frac{F_{ij}}{d_{ij}}} \tag{4}$$

The algorithm assumes:

1. The nodes of the network and the input traffic flow for each pair of nodes are known.

2. A routing model determines the optimal flows F_{ij} of all links (i, j) given the link original capacities C_{ij}. We assume that the link flows minimize a cost function $\sum\limits_{ij} D_{ij}(F_{ij})$ as in Eq.(4.9). F_{ij} can be determined by minimizing the average packet delay,

$$T = \frac{1}{\gamma} \sum_{(i,j)} \left(\frac{F_{ij}}{C_{ij} - F_{ij}} + F_{ij}p'_i \right)$$

based on the $M/M/1$ formula, where γ is the total input traffic into the network, and C_{ij} and p'_i are the capacity and the processing and propagation delay, respectively, of link (i, j). The algorithms described in Sections 4.2, 4.3 and 4.4 can be used for this purpose.

5.2 The algorithm

This section describes the different steps of the capacity assignment algorithm and how it works.

The capacity assignment algorithm

Step 1: Select a network topology with initial capacities and requirements.

Step 2: Compute optimal link flows that minimize the average delay for the network using the EFDA algorithm.

Step 3: Allocate the link capacities to minimize the delay with the link flows computed in step 2, given the constraints on the total cost of a system.

Step 4: Use these capacities instead of the original capacities with the original requirements and go to step 2. The delay calculated in this step will be less than in step 2.

Step 5: Reallocate the optimal link capacities with the optimal link flows from step 4. The new delay will be smaller than the delay in step 3.

The iteration is repeated until the new network delay is not significantly smaller than the old delay. Since the delay decreases with every iteration and it is positive, the algorithm converges.

In order to evaluate the effectiveness of the above method, the algorithm was applied to a model of a network consisting of 8-nodes and 10-links presented in [2] to compute the optimal link capacities. The topology of the network is shown in Fig.5.1. Each link carries traffic in both directions. The double lines between nodes 3 and 4 and 7 and 8 indicate that there are two links in each direction connecting theses nodes. The network carries 2 traffic classes: the bandwidth requirement of the 1st class is 1 unit and the bandwidth requirement of class 2 is 40 units. The capacity of each link is 5624 units. The traffic intensity matrix is given in table 5.1 and the class dependent intensities are given by $\rho_{ij}^{s} = \rho_{ij}\gamma_s b_s$ where ρ_{ij} represents the traffic intensity between link (i, j), γ_s is the class load factor and b_s the bandwidth. These value are given in table 5.2.

We now compare the network optimal link capacities with their initial values. Table 5.3 shows the optimal link flows and capacities computed after the convergence of the capacity assignment algorithm.

Fig 5.2 plots the capacity assigment for link (1-2) as the algorithm executes. Fig 5.3 plots the network delay as the algorithm executes. The delays converge after 44 iterations. Comparison with Fig 5.2 shows a strong correlation between the delay and the assigned capacity. This can be expected since the delay is a function of capacity. The delays decrease since the flows are shifted onto optimal paths in order to reduce

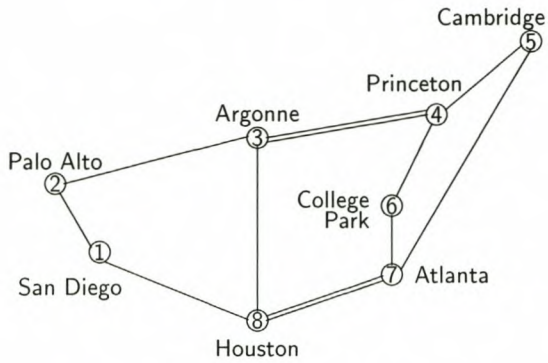


Figure 5.1: The Core NSF ATM network

nodes	1	2	3	4	5	6	7	8
1	–	13	15	2	20	10	4	6
2	–	–	49	6	64	29	11	17
3	–	–	–	7	76	34	13	21
4	–	–	–	–	9	4	2	2
5	–	–	–	–	–	45	17	27
6	–	–	–	–	–	–	8	12
7	–	–	–	–	–	–	–	4
8	–	–	–	–	–	–	–	–

Table 5.1: The NSF network: traffic intensity matrix

the delays on congested links. This smaller flow on the congested links in turn leads to a decrease in the capacity required to achieve a given delay.

There are several approaches to capacity assignment, utilizing different performance criteria. An additional approach is to improve an existing network by redistributing the link capacities while maintaining the total sum of all capacities of the network.

Kleinrock [4] notes that the selection of an appropriate algorithm to allocate capacities will depend on the cost-capacity structure, on the presence of additional topological

	class 1	class 2
γ_s	20	0.1
b_s	1	40

Table 5.2: Load factor and slots per service

Links	Optimal Flows	Optimal Cap	Initial Cap
(1,2)	520	1469.4	5624
(1,8)	2280	4268.1	5624
(2,3)	7040	10533.4	5624
(3,4)	8800	12705.8	11248
(3,8)	3240	5609.9	5624
(4,6)	4480	7266.8	5624
(5,7)	2640	4779.3	5624
(6,7)	1200	2642.3	5624
(7,5)	2640	4779.3	5624
(7,8)	4040	6686.4	11248

Table 5.3: Optimal link flows and link capacities

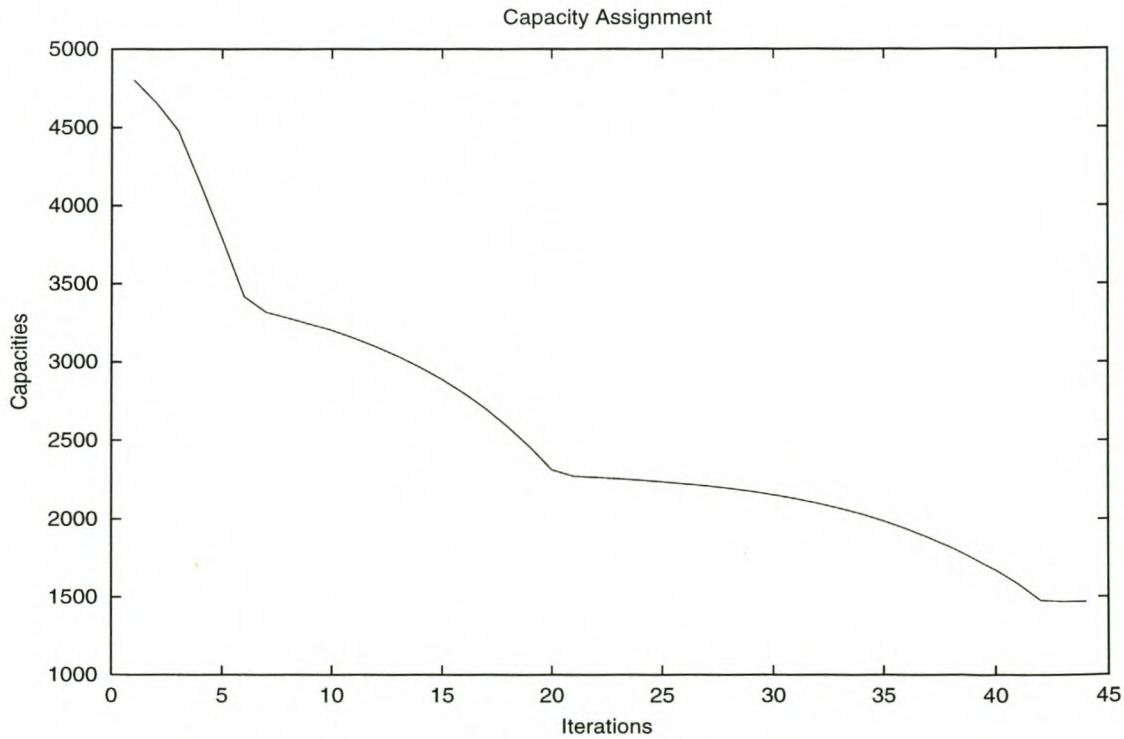


Figure 5.2: Capacity assignment algorithm

constraints, on the degree of human interaction allowed and, finally, on the tradeoff between cost and precision required by the particular application.

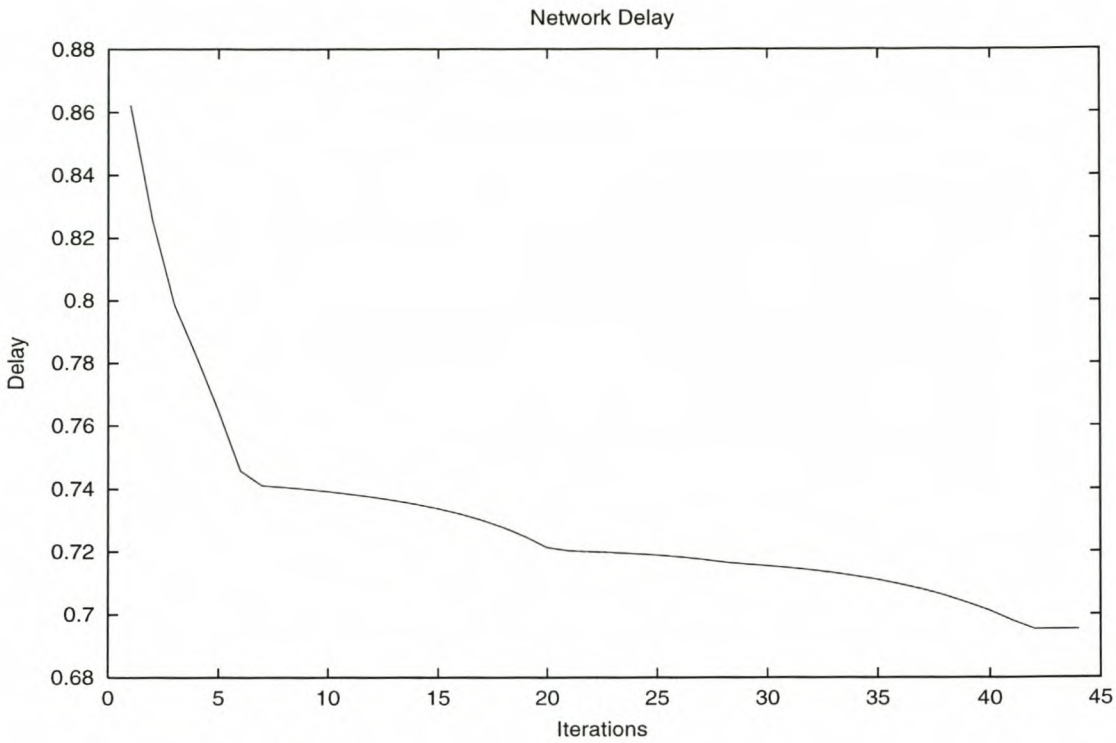


Figure 5.3: Convergence of the algorithm

5.3 Multiservice Blocking Model

In this section we again model a network as a collection of resources to which calls, each with an associated holding time and class, arrive at random instances. However, this time a call can be blocked.

Recall that the EFDA uses an objective function based on the $M/M/1$ queue. In this system, if a message or customer arrives when the channel is not busy, no message in transmission, the message is transmitted immediately. If the channel is busy when the message arrives, the message is placed in a queue where it waits until the channel becomes free and then begins serving the next message.

Not all the systems deal with congestion by allowing messages to wait. Most traditional telephone systems block calls from entering the system if no capacity is available for them.

This kind of system is fundamentally different from a queueing system because a call's system time is equal to its holding time. Here, a call arrives and requires a fixed amount of capacity, enough to handle a conversation. If the capacity is available, it is

dedicated to the call for its duration. If not the call is blocked and lost. We consider such a system in this section.

5.4 Network model

In our blocking model, a network consists of N nodes with L physical links which carries S classes of calls. Each link $i-j$ has capacity of C_{ij} bandwidth units. We define a route as a sequence of physical links. Let $\mathcal{R}_{ij}$ denote the set of routes that connect i and j. Class-s calls are offered to O-D pair (i,j) according to a Poisson process with rate $\lambda^s_{i,j}$. The average holding time of a call of class-s is exponentionally distributed with mean $1/\mu^s_{i,j}$. We refer to $\rho^s_{i,j} = \lambda^s_{i,j}/\mu^s_{i,j}$ as the class-s intensity of the offered traffic stream. The bandwidth requirement of a class-s call is b_s. A class-s connection between O-D pair (i,j) is admitted if there is sufficient bandwidth available on at least one route in $\mathcal{R}^s_{ij}$ to accommodate its effective bandwidth and is lost otherwise.

Our goal is to obtain a capacity assignment such that the link blocking probabilities satisfy a certain grade of service (GOS).

5.5 Implementation

We compute the link blocking probabilities B^s_{ij} as

$$B^s_{ij} = E^s_{ij}(\rho_{ij}, \mathbf{b}, C_{ij}) \tag{5}$$

where $\rho_{ij} = (\rho^s_{ij})_{s\in S}, \mathbf{b} = (b_s)_{s\in S}$ and $E^s_{ij}(.)$ is a function returning the blocking probabilities of class-s calls on link (i,j).

5.5.1 Blocking probabilities

There are several options for the blocking function $E^s_{ij}(.)$. We use the stochastic knapsack algorithm [8].

The implementation starts by computing link flows that are optimal in terms of the network delay. Then we compute the link blocking probabilities using these flows for each link. To calculate the probabilities we need the traffic intensities for different classes which are given by $\rho^s_{ij} = \rho_{ij}\gamma_s b_s$ where ρ_{ij} represents the traffic intensity on

Links	Service Integration		Service Separation	
	class1 $\times 10^{-4}$	class2 $\times 10^{-3}$	class1 $\times 10^{-3}$	class2 $\times 10^{-3}$
(1,2)	2.04698	9.99201	9.78443	9.69459
(1,8)	2.13866	9.95198	9.87183	9.22791
(2,3)	2.15838	9.99911	9.96748	9.00554
(3,4)	2.22972	9.96107	9.96809	8.91993
(3,8)	1.97635	9.95442	9.67632	6.76622
(4,5)	2.16964	9.98509	9.89086	9.74935
(4,6)	2.04468	9.97948	9.86695	9.85335
(5,7)	2.15711	9.93149	9.88864	9.41679
(6,7)	2.04277	9.92063	9.79374	7.07872
(7,8)	2.16638	9.96009	9.85324	8.57969

Table 5.4: The link blocking probabilities per service class

link (i, j), γ_s is the class load factor and b_s the bandwidth. After calculating these intensities, we run the multiservice blocking probability algorithm. The algorithm uses the link capacity as a loop index. Thus we first calculate blocking probabilities for the link with capacity 1, then use this to calculate for link with capacity 2, and so on. When the capacities are big enough, the blocking probability becomes very small, almost zero. The iteration terminates as soon as the average blocking probability becomes less than the GOS.

5.5.2 Service Separation

Under service integration, all the bandwidth C_{ij} of link (i, j) is available to all service classes. With service separation, each class has access only to a different bandwidth $C_{ij}^1, C_{ij}^2, \cdots, Cij^S$ where $C_{ij}^1 + C_{ij}^2 + \cdots + C_{ij}^S = C_{ij}$.

In this case, we calculate the optimal link capacities required for every class with certain blocking probability, then we sum all capacities to obtain the capacity on a link. Consider again the network model in section 5.2. The network consists of 8 nodes and carries 2 traffic classes. The traffic intensity matrix is given in table 5.1 and the class dependent intensities are given by multiplying these values by the load factors.

Table 5.4 presents the link blocking probabilities per service class for service integration as well as service separation and we compare the link capacities produced by the two services in table 5.5.

Links	$\mathbf{F}_{i,j}^{s}$		$\mathbf{Cap}_{int}$	$\mathbf{Cap}_{sep}$
	class1	*class2*		
(1,2)	1408.4	845.1	2671	2678
(1,8)	2273.1	1363.8	4130	4140
(2,3)	2386.9	1422.9	4320	4333
(3,4)	3834.1	2300.4	6723	6732
(3,8)	1085.1	651.1	2118	2155
(4,5)	2581.5	1548.9	4645	4647
(4,6)	1410.7	846.4	2675	2680
(5,7)	2574.4	1544.7	4634	4640
(6,7)	1478.5	887.1	2791	2828
(7,8)	2615.4	1569.3	4702	4721

Table 5.5: Optimal link flows and capacities per service class

We can see from the Table 5.5 that links with larger flows have larger capacities and therefore smaller service times which is a factor contributing to the reduction in the total delay of the network.

Note that our experiment combines two different models: the blocking model and the queueing model. However, these models are applied at different stages. First we have the queueing model associated with a network delay. Secondly, we have the blocking model associated with calls blocked and packets dropped. In first case, the capacities must be enough to accomodate the optimal flows, otherwise the network delay becomes indefinite. In the second situation, some capacities might be insufficient and calls will be dropped.

In order to evaluate the effect of applying service separation, we also compute the optimal link capacities for service integration and then compare the two. The total link capacity computed in service separation is the sum of the capacities required in both traffic classes.

Our conclusion is that the new capacity is sometimes bigger than in the case of complete sharing; this is the penalty for service separation; the penalty is not too large. For example, for link (4,5) the link capacity on complete sharing is 4645 while the separation case is 4647.

Chapter 6

EFDA: Numerical Results

In order to test the performance of the extended flow deviation algorithm under different conditions its performance was tested on several topologies and for different parameter values. Different topologies were used in the experiments but we will analyze two of them in detail in the following section. Some data for these tests networks were extracted from [22]

6.1 Fifty-Node Test Network

We apply the EFDA algorithm to a larger network, also used by C. Villamizar [22] with a significantly larger number of OD pairs. The topology of the network is shown in Fig 6.1. This network has fifty nodes and 202 links. All links are uni-directional and have different bandwidths.

The results of the experiments are described by providing the values of the optimal routes, the optimal solution, the optimal link capacities and the average end to end delay in the network corresponding to the best feasible solution.

In order to evaluate the effectiveness of the algorithm, different parameter values were applied to the network and the results compared.

The set of original link capacities is given in the second plot of Fig 6.4. The results of the first experiment presented in Fig 6.2 show the optimal flows for the EFDA algorithm and MPLS-OMP data. The results are generated after 53 iterations. Execution time for the EFDA algorithm was less than 1min on Pentium II 200 MHz. The convergence

of the algorithm is showing in Fig 6.3.

The Fig 6.2 also presents the comparison between the EFDA and the MPLS-OMP method. The plot 6.2 shows the link loads as computed by the EFDA algorithm and from the MPLS-OMP data. Both figures represent the optimal solution generated in those experiments.

The first plot displays the link loads generated by the EFDA algorithm. The average link load was computed to be 50 %. The highest link loads are 71.17 %, 67.65 %, 67.51 %, 64.92 % We also have 0 % on link (23, 1). The link capacity of (23, 1) is 28165 bits/sec while the link capacity of (23, 20) is 237765 bits/sec and link (20, 1) is 339945 bits/sec. The shortest path is (23 $\Rightarrow$ 20 $\Rightarrow$ 1) and not (23 $\Rightarrow$ 1). The (23, 1) link was thus never selected as an appropriate path.

The second plot depicts the link loads by MPLS-OMP after the first 10 minutes of convergence. Convergence is essentially complete at 60 minutes. The worst link loads are less than 70 % and the average link load is 60 %. The two plots are superimposed to see the difference of the link loads of the two methods in Fig 6.2. Our flow deviation algorithm spreads the traffic in the network in such a way that the link loads are lower than in the case of the OMP method. Therefore figure 6.2 shows that flow deviation's link load percentages are in general lower than OMP's.

Another experiment involved computing the optimal capacities for the initial configuration and then evaluating the performance of the network. Fig 6.4. demonstrates the results: the algorithm allocates more capacity to the links with more flow and therefore provides smaller service times which is a factor contributing to the reduction in delay. The two plots are superimposed in Fig 6.4. As the plot shows, the difference between the optimal and the original capacities is not large. The pattern of the link capacities is the same. Some of the solutions are identical and the narrow range of capacities implies that it is close to the optimum.

One can see the convergence of the algorithm and how the network delay gets better and better after each iteration in Fig 6.3.

6.2 A Hundred-Node Test Network

The EFDA algorithm was applied to investigate the optimal link flows and link capacities for a network with hundred nodes and 244 links. The network topology is

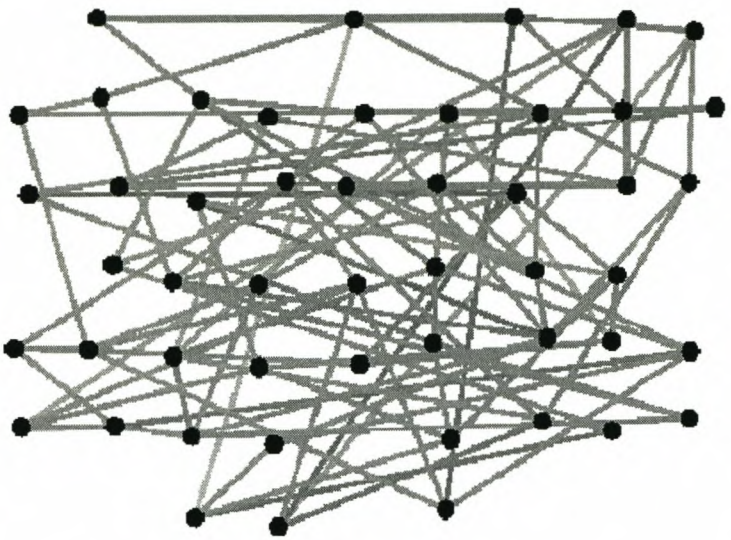


Figure 6.1: MPLS-OMP network topology

presented in Fig 6.5. The links are uni-directional.

Fig 6.6 and 6.7 summarize the results of an optimization minimizing the network delay. Only 10 iterations were required for convergence. As shown in Fig 6.6, the link capacities tend to increase as the algorithm proceeds until any further increase would not significantly decrease the delay.

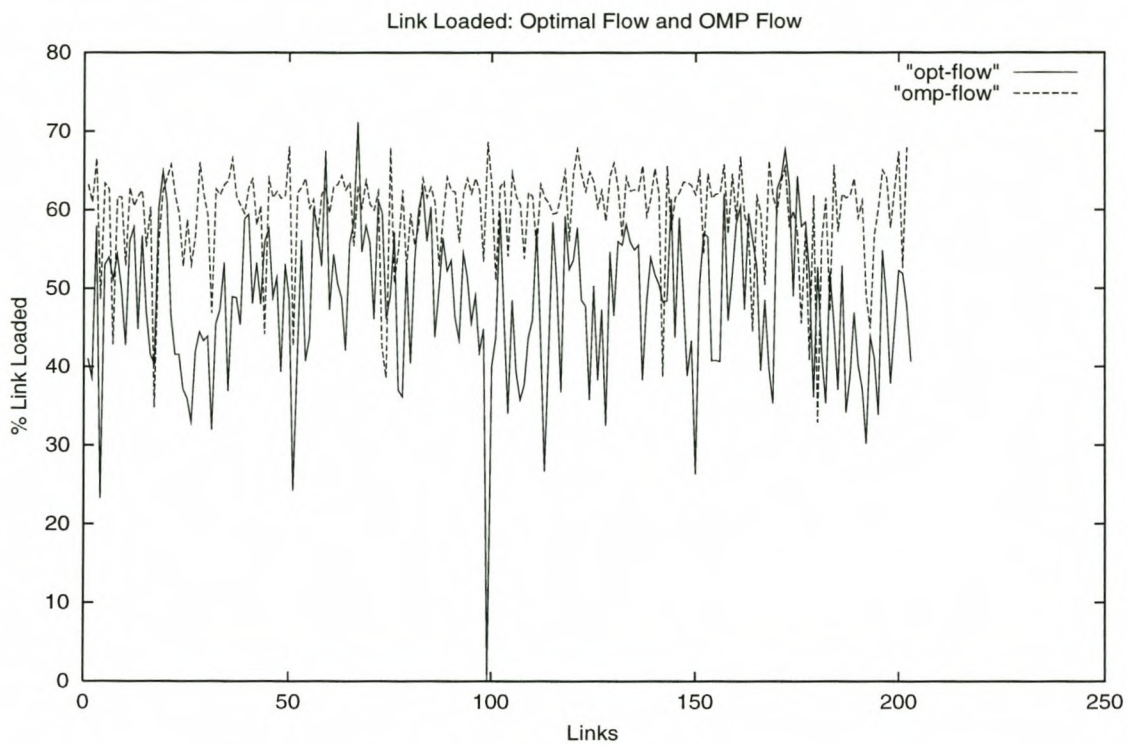


Figure 6.2: The link load

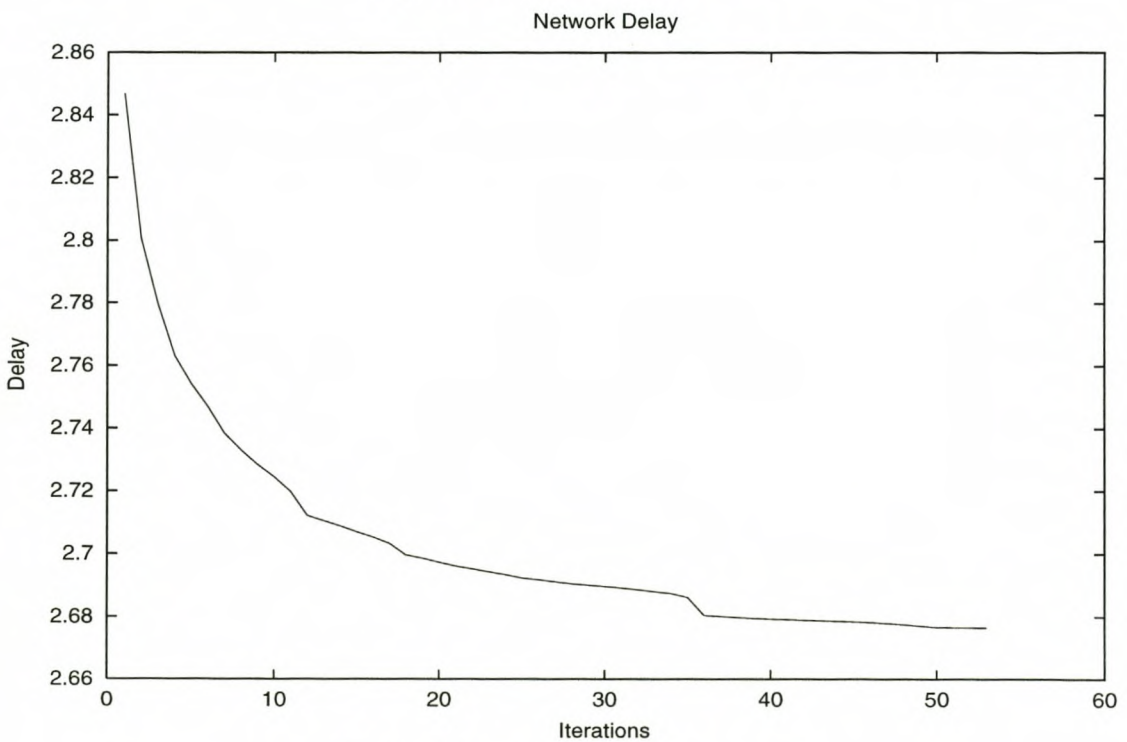


Figure 6.3: The network delay

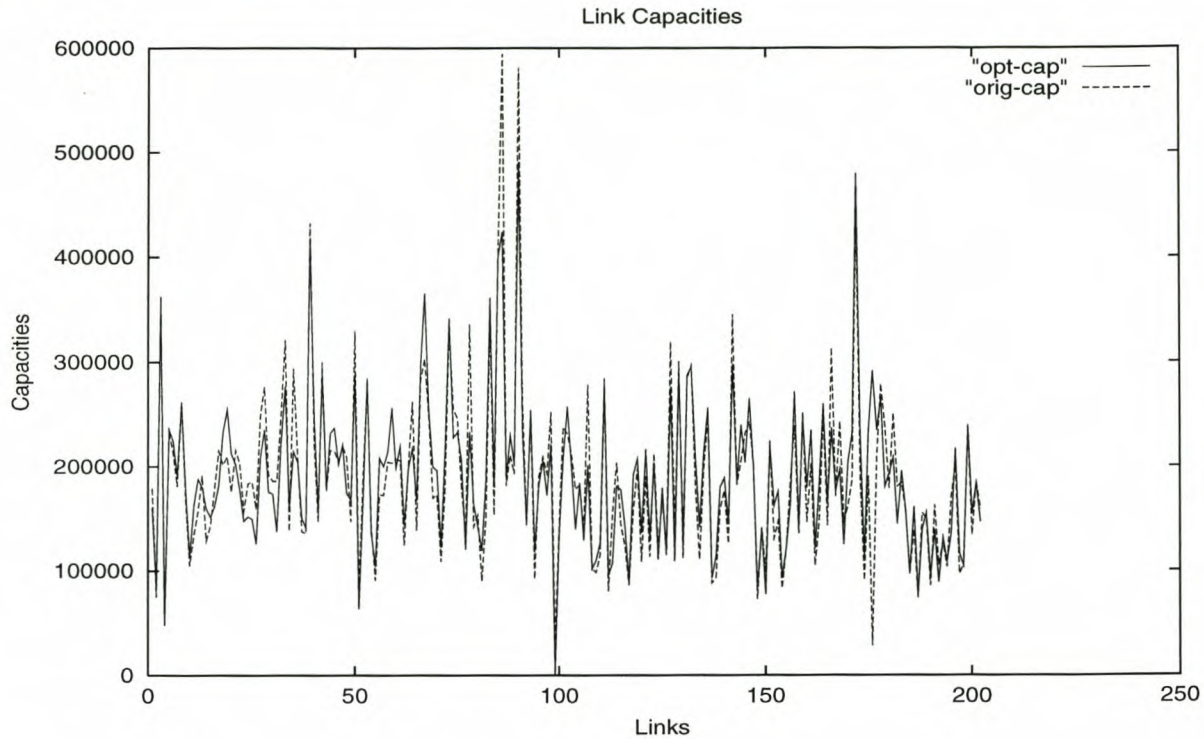


Figure 6.4: The link capacities

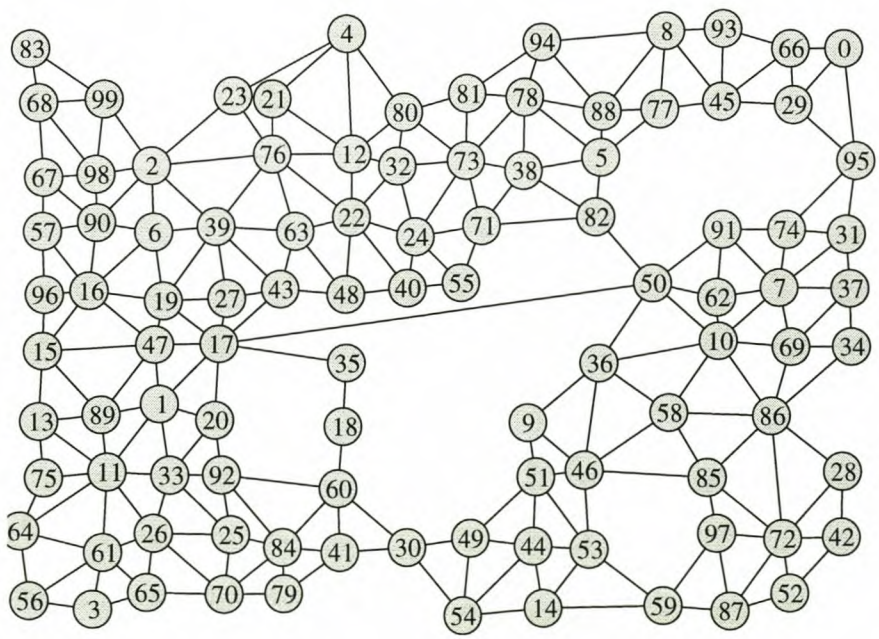


Figure 6.5: Hundred-Node Network

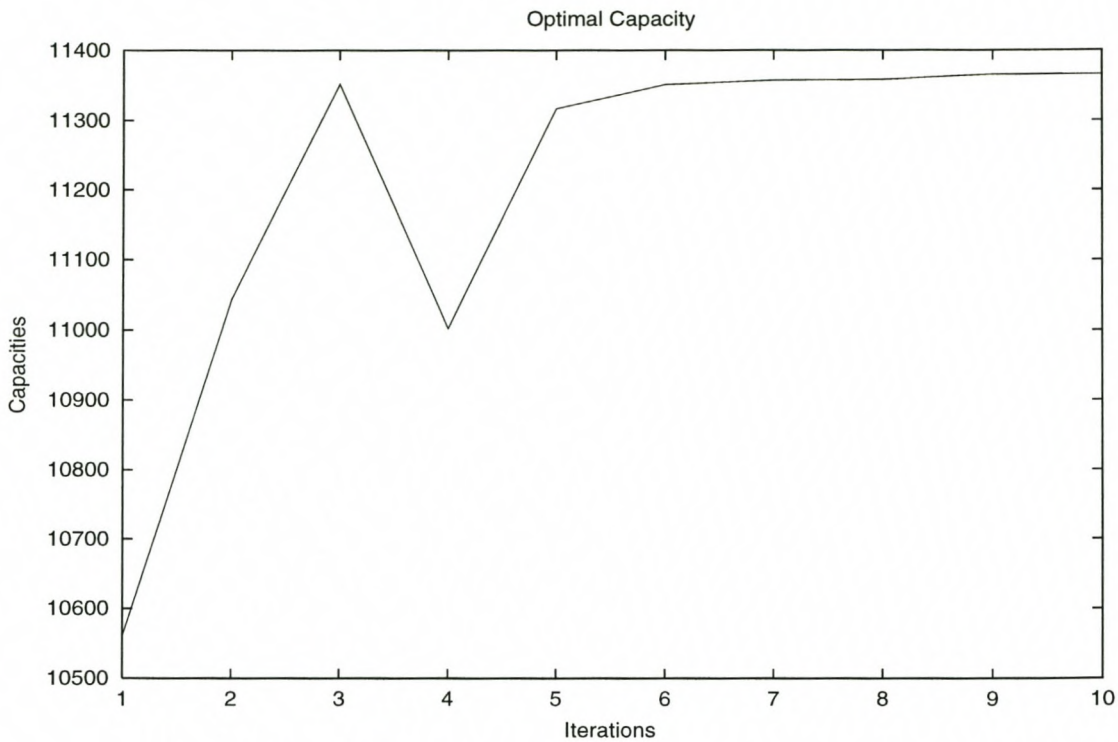


Figure 6.6: The optimal capacities

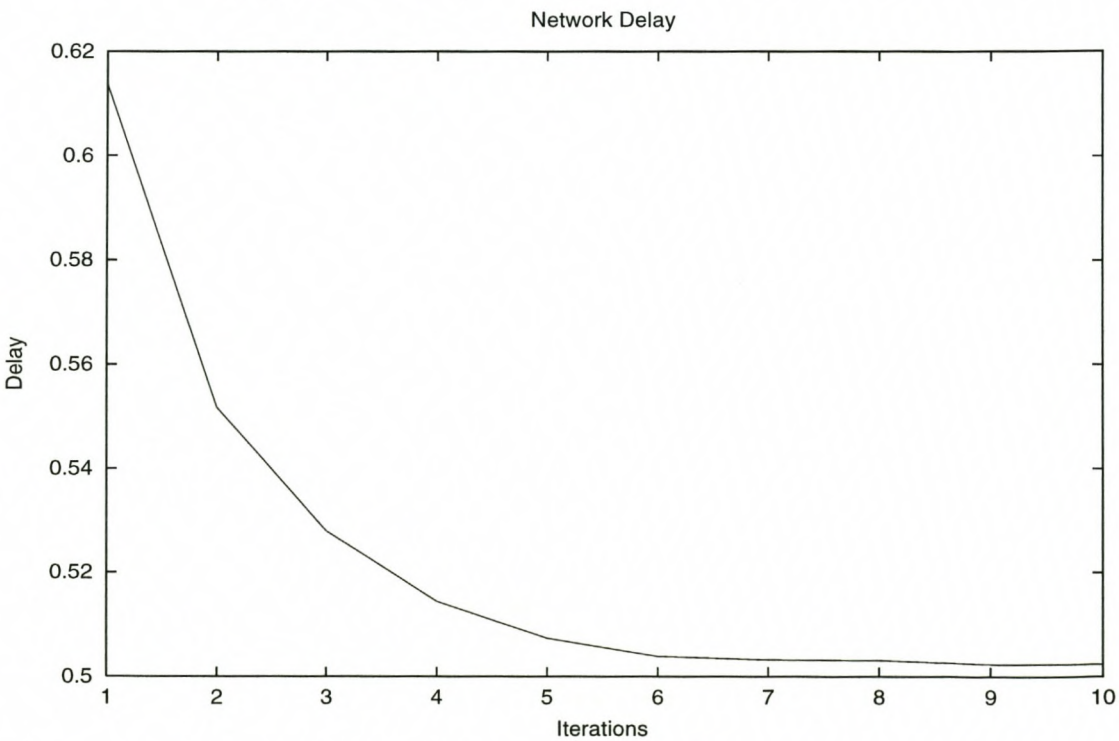


Figure 6.7: Hundred-Node network delay

Chapter 7

Conclusions

This thesis surveyed some of the mathematical programming and network routing (flow) techniques that have been found to be useful for the design of the computer-communication networks.

Our main focus was on methods for optimal routing which can be subdivided into path discovery and packet forwarding.

We present an algorithm which combines flow deviation with Lagrangian multiplier relaxation in order to compute efficiently the shortest routes, assign optimal flows to these routes and simultaneously to compute optimal link capacities in both single and multirate networks.

Applying the results to a set of networks, we show that the algorithm converges to a good solution faster and generates better feasible solutions than the Bertsekas-Gallager flow deviation algorithm.

Appendix A

The Blocking Probability

A.1 The Erlang-B Formula

Erlang's function

$$B(\rho, C) = \frac{\rho^C / C!}{\sum_{c=0}^{C} \rho^c / C!} \tag{1}$$

is perhaps the single most important expression in the field of teletraffic theory. It was derived by the A.K. Erlang as a formula for the probability that calls arriving individually in a Poisson stream of intensity ρ to a link consisting of C circuits would find all circuits occupied and therefore be lost. Erlang's function is mostly used in the field of telecommunications, in its own right and as part of more complicated analysis procedures, both exact and approximate. The function is numerically difficult to evaluate because it involves a very large number of C. Fortunately, it, and its related functions such as its derivatives can be rearranged algebraically into a more convenient form which allows us to iteratively compute $B(\rho, C)$ as in the following recursion

$$B(\rho, C+1) = \frac{\rho B(\rho, C)}{\rho B(\rho, C) + C + 1} \tag{2}$$

This equation has computation requirements $O(C)$ and storage requirements $O(1)$. The recursion works with normalized quantities $B(\rho, C)$ and is therefore not subject to numerical problems such as imprecision and overflow.

A.2 Multi service Blocking Probability

The classical deterministic knapsack problem (see Ross [5,23] for example) involves a knapsack into which objects from K different classes arrive at random and share C resource units. An object departs after its holding time completes. The stochastic system can be used to model a link in a multi service telecommunication technology and calculate the blocking probabilities of all the traffic classes sharing the link.

Consider a stochastic knapsack with capacity of C resource units to which objects from K classes arrive. Let λ_k denote the Poisson arrival rate of a class-k object and $1/\mu_k$ the average holding time of this class-k object. If an arriving class-k object is admitted into the knapsack, it holds b_k resource units and departs at the end of this holding time, all the b_k resource units are simultaneously released. Let $\rho_k = \lambda_k/\mu_k$ and let $B_k(C)$ denote the blocking probability of class-k objects for this knapsack.

Berezner and Krzesinski [8] have developed the following numerically stable algorithm to compute the multiservice blocking probabilities for the basic stochastic knapsack:

1. $M = b_K$
2. $P(M) = 1$
3. $P(m-1) = 0 \qquad m = 2, \ldots, M$
4. **for** $(c = 1; c <= C; c++)$ {
   $\alpha = (1/c) \sum_{k=1}^{K} \rho_k b_k P(M + 1 - b_k)$
   $P(m-1) = P(m)/(1+\alpha) \qquad m = 2, \ldots, M$
   $P(M) = \alpha/(1+\alpha)$
   }
5. $B_k(C) = \sum_{m=0}^{b_k - 1} P(M-m) \qquad k = 1, \ldots, K$

Figure 1. Algorithm to compute the Erlang-B blocking probabilities $B_k(C)$

The general stochastic knapsack, which can be also used for modeling networks, differs from the basic stochastic knapsack in that the arrival rates $\lambda_k(\mathbf{n})$ and the mean holding times $1/\mu_k(\mathbf{n})$ or state-dependent. These parameters dependent on the state $\mathbf{n} = (n_1, n_2, \ldots, n_K)$ where n_k is the number of class-k objects in the knapsack, of the knapsack.

Bibliography

[1] A. Kershenbaum. Telecommunications Network Design Algorithms. McGRAW-Hill International Computer Science series, 1993.

[2] D. Mitra, J.A. Morrison and K.G. Ramakrishnan. ATM network design and optimization: A multirate loss network framework. *In Proceeding of IEEE INFOCOM'96*, vol.3 pages 994-1002, San Francisco, USA, March 1996.

[3] D. Bertsekas and R. Gallager. Data Networks, Prentice-Hall, Englewood Cliffs, NJ 1992.

[4] L. Fratta, M. Gerla and L. Kleinrock. "The flow deviation method: An approach to store and forward communication network design" Networks, 3:97-133, 1990.

[5] K.W. Ross. Multiservice Loss Models for Broadband Telecommunication Networks. Springer-Verlag, London, 1995.

[6] COST 242. Broadband Network Teletraffic: Performance Evaluation and Design of Broadband Multiservice Network. 1996.

[7] E.S. Levitin, and B.T. Poljak. Constrained Minimization Methods, *URSS Computer Math. Phys.*, 6:1-50. 1985.

[8] S.A. Berezner and A.E. Krzesinski. An Efficient Stable Recursion to Compute Multiservice Blocking Probabilties. Submitted for publication.

[9] B. Davies, P. Doolan and Y. Rekter. Switching in IP Networks: IP Switching, Tag Switching, and Related Technologies. Morgan Kaufmann Publishers, Inc. San Francisco, California.

[10] M. Frank, and P. Wolfe. "An algorithm for quadratic programming." Naval Res. Logist. Quart., 3:149-154, 1991.

[11] L. Kleinrock. Queueing System, Vol. 2: Computer Applications. John Wiley Sons, New York, 1976.

[12] E.D. Lazowska, J. Zahorjan, G.S. Graham and K.C. Sevcik. Quantitative System Performance. Computer System Analysis Using Queueing Network Models. Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1984.

[13] K.S. Vasta, *A Numerical Study of two measures of Delay for Network Routing,* M.S.thesis, University of Illinois, Dept. of Electrical Engineering, Urbana, IL. 1979.

[14] R.K. Ahuja, T.L. Magnanti and J.B. Orlin. Network Flows Theory, Algorithms and Applications Prentice-Hall, Englewood Cliffs, New Jersey, 1993.

[15] W.H. Press, B.P. Flannery, S.A. Teukolsky and W.T. Vetterling. Numerical Recipes in C. The Art of Scientific Computing. Cambridge University Press, 1988.

[16] U. Manber. Introduction to Algorithms A creative approach Addison-Wesley 1989.

[17] F. Kelly, "Blocking Probabilities in Large Circuit Switched Networks", Adv. Appl. Prob., 18 (1986), 473-505.

[18] A.O. Allen. Probability, Statistics, and Queueing Theory: with Computer Science Applications. Academic Press, Inc., 1998.

[19] N. Anerousis and A.A. Lazar. Virtual Path Control for ATM Networks with Call Level Quality of Service Guaranees. *IEEE/ACM Transactions on Networking.* vol.6 1998.

[20] A. Inggsi. Call Management in Broadband Networks. Technical Report Department of Computer Science, University of Stellenbosch, 7600 Stellenbosch, South Africa 1999.

[21] J.S. Kaufman. Blocking in a shared resource environment. *IEEE Transactions on Communications*, COM-29(10):1474-1481, October 1981.

[22] C. Villamizar. MPLS Optimized Multipath (MPLS-OMP), internet draft-ietf-ospf-omp-03 August 18, 1999.

[23] K.W. Ross and D.H.K. Tsang. The Stochastic Knapsack Problem. *IEEE Transactions on Communications* 37:7, pp 740-747, 1989.

[24] D. Awduche, J. Malcolm, J. Agogbua, M. O'Dell, J. McManus. Requirements for Traffic Engineering over MPLS. draft-ietf-RFC 2702 September, 1999.

[25] C. Metz. IP Switching Protocols and Architectures, McGraw-Hill, 1999.

[26] Rekhter, et al. Switching in IP Networks, Morgan Kaufmann, 1998.

[27] Feldman, et al. "Evolution of Multiprotocol Label Switching" *IEEE Communications*, vol.36 No.5 May 1998.

[28] G. Apostolopoulos et al. QOS Routing Mechanisms and OSPF Extensions. draft-ietf-RFC 2676 August 1999.

[29] E.C. Rosen, A. Viswanathan and R. Callon. Multiprotocol Label Switching Architecture. draft -ietf-mpls-arch July 2000.