Path selection and bandwidth allocation in MPLS networks: a non-linear programming approach

James E Burns[a], Teunis J Ott[a], Johan M de Kock[b] and Anthony E Krzesinski[b]

[a]Telcordia Technologies Inc, 445 South Street, Morristown NJ 07960-6438, USA
[b] Department of Computer Science, University of Stellenbosch, 7600 Stellenbosch, South Africa

ABSTRACT

Multi-protocol Label Switching extends the IPv4 destination-based routing protocols to provide new and scalable routing capabilities in connectionless networks using relatively simple packet forwarding mechanisms. MPLS networks carry traffic on virtual connections called label switched paths. This paper considers path selection and bandwidth allocation in MPLS networks in order to optimize the network quality of service. The optimization is based upon the minimization of a non-linear objective function which under light load simplifies to OSPF routing with link metrics equal to the link propagation delays. The behavior under heavy load depends on the choice of certain parameters: It can essentially be made to minimize maximal expected utilization, or to maximize minimal expected weighted slacks (both over all links). Under certain circumstances it can be made to minimize the probability that a link has an instantaneous offered load larger than its transmission capacity. We present a model of an MPLS network and an algorithm to find and capacitate optimal LSPs. The algorithm is an improvement of the well-known flow deviation non-linear programming method. The algorithm is applied to compute optimal LSPs for several test networks carrying a single traffic class.

Keywords: Internet Protocol, label switched path, multi-protocol label switching, quality of service

1. INTRODUCTION

The Internet is becoming the ideal platform to support all forms of modern communications including voice, data and multimedia transmissions. However, the standard IPv4 routing protocols were developed on the basis of a connectionless model where each packet's next hop is determined by an computationally expensive longest-prefix-match mechanism and routing decisions are based on simple metrics such as hop-count which leads to the selection of shortest path routes. Despite its ability to scale to very large networks, this approach provides only rudimentary Quality of Service (QoS) capabilities which cannot be used to provide scalable service level agreements for bandwidth intensive applications in modern networks.

Multi-protocol label switching (MPLS)[1] extends the IPv4 destination-based routing protocols to provide new and scalable routing capabilities. MPLS routing/switching is achieved by forwarding IP packets along virtual connections called label switched paths (LSPs). LSPs are set up by a label distribution mechanism which uses the information contained in layer 3 routing tables. The LSPs form a logical network that is layered on top of the physical network to provide connection oriented processing above the connectionless IP network.

In this paper we formulate the problem of finding an optimal set of LSPs and optimally allocating bandwidths to these LSPs as a constrained non-linear programming problem (NLP) which minimizes an appropriate objective function. In qualitative terms the goal is to find a set of LSPs and a set of target bandwidths for these LSPs such that if the traffic forecasts are exact and all target bandwidths of LSPs are achieved, the system will carry all the offered traffic, no link is too heavily utilized, and the carried load is appropriately distributed.

Several previous studies (see[2–4] and the references therein) have formulated the bandwidth allocation problem in connectionless networks as an NLP using the M/M/1 formula as an objective function to predict the queueing delay on individual links, and a load balancing scheme is considered optimal if it minimizes the total delay over the network. However, the delay in an internet is limited by the drain time of buffers. Furthermore, TCP congestion

Further author information: (Send correspondence to A.E.K.)
J.E.B. and T.J.O.: Email: {burns,tjo}@research.telcordia.com J.M.K. and A.E.K.: Email: {jmdekock,aek1}@cs.sun.ac.za

avoidance and RED (Random Early Discard) schemes (see[5,6] and the references therein) make it possible to have a very high sustained utilization on a link with simultaneously only moderate packet loss and only moderate variability in buffer occupation. The use of an M/M/1 queueing delay in the objective function is therefore highly suspect or even incorrect. The issue is not only that the M/M/1 formula is poor in predicting actual queueing delay, but that queueing delay is moderately insensitive to traffic intensity on a link. Mechanisms like weighted fair queueing (WFQ) and class-based WFQ will make queueing delay even more independent of link utilizations.

Our approach to the NLP and its solution has several novel aspects. First, we present an objective function that affords an appropriate representation of the actual quality of the network: under light load our objective function simplifies to OSPF routing with link metrics equal to the link propagation delays. The behavior under heavy load depends on the choice of certain parameters: It can essentially be made to minimize maximal expected utilization, or to maximize minimal expected weighted slacks (both over all links). Under certain circumstances it can be made to minimize the probability that a link has an instantaneous offered load larger than its transmission capacity.

Second, we present an efficient technique to solve the NLP. We have adapted an existing solution technique, namely the flow deviation method[2–4] to optimize our objective function. Our implementation of the flow deviation algorithm differs from the standard method in that we identify a working set of LSPs and re-distribute bandwidth over these LSPs until it becomes advantageous to admit new LSPs to the working set. Appropriate numerical methods and data structures are used to achieve an efficient implementation of the NLP solver. The advantage of our flow deviation method is that, for the objective functions we use, our flow deviation method is several times faster than the standard flow deviation method.

The flow deviation algorithm computes optimal flows in connectionless networks: the objective function is a sum of link revenue functions and the flow is optimally allocated among the links and routes. Recently much attention has been given to adapting optimization algorithms, which were originally developed for circuit switched operations, so that they can be applied to logically fully connected networks which are layered on top of connectionless networks. For example, the capacity routing algorithm (see[7] and the references therein) discovers and capacitates optimal routes in multi-service connection-oriented networks. Here the objective function expresses an end-to-end service measure such as the call blocking probability. Bandwidth is optimally allocated among the discovered routes to form virtual path connections which are either shared among the traffic classes (service integration) or separate VPCs are allocated to each service class (service separation).

The rest of the paper is organized as follows. Section 2 presents a model of an MPLS network, definitions of feasible and optimal LSP bandwidth assignments, a description of the LSP design problem whose solution yields an optimal set of LSPs and optimal LSP bandwidth assignments, and a description of a penalty function which under light load simplifies to OSPF routing and depending on certain parameter choices under heavy load optimizes one of a range of performance criteria. Section 3 describes our implementation of the flow-deviation algorithm to solve the LSP design problem. Section 4 applies the flow deviation method to compute optimal LSP sets for models of 50- and 100-node networks. The characteristics of the LSP sets are described. Our conclusions are presented in Sect. 5.

2. THE MODEL

Consider a communications network with N nodes and L links. Let $\mathcal{N} = \{1, 2, \ldots, N\}$ denote the set of nodes and let $\mathcal{L} = \{1, 2, \ldots, L\}$ denote the set of links. The nodes represent the routers in the MPLS-capable core of a network. Some nodes are connected by a link. The links are directed: each link has a starting node and an ending node which are routers from the set $\mathcal{N}$.

Each node $m \in \mathcal{N}$ is both an ingress router and an egress router. Each node is an ingress router because traffic from the non MPLS-capable part of the network enters the MPLS network at that point. Each node is an egress router because traffic to the non MPLS-capable part of the network exits from the MPLS network at that point.

Let $d_{(m,n)}$ denote the predicted demand (offered load) of traffic that wants to enter the MPLS network at node m and wants to exit at node n. We assume that the demands $d_{(m,n)}$ and the link capacities b_i are such that a feasible solution exists. The definition of feasibility will be given shortly. If a feasible solution does not exist then systematic drop (discard) of traffic is necessary, and it is an interesting question what traffic needs to be dropped to minimize the damage. We consider only a single class of service.

2.1. Paths and Path Bandwidths

A path P is a sequence of links $L_1, L_2, \ldots, L_{H_P}$ where $H_P \geq 1$ is the *hop count* of the path P. In our terminology a route and a path and an LSP (label switched path) are synonymous. No path traverses the same link or the same node more than once. The algorithms that follow in later sections ensure that no paths contain cycles. Let $\mathcal{P}$ denote the set of all such non-cycling paths. Since any path P contains no cycles, the sequence of links traversed by a path P can be interpreted as a set denoted by $\mathcal{L}_P$. Let $\mathcal{P}^{(i)}$ denote the set of paths that utilize link i. Let $\mathcal{P}_{(m,n)}$ denote the set of paths from node m to node n with $m \neq n$.

2.2. Feasibility and Optimality

Each path P will be assigned a *target bandwidth* $B_P \geq 0$. The goal is to select these target bandwidths in an in some sense optimal way. Let $\boldsymbol{B} = (B_P)_{P \in \mathcal{P}}$ denote a set of target bandwidths. $\boldsymbol{B}$ is said to be feasible if the following two constraints hold:

1. For each pair of nodes (m, n)

$$\sum_{P \in \mathcal{P}_{(m,n)}} B_P = d_{(m,n)} \tag{1}$$

   so that if the traffic forecasts $d_{(m,n)}$ are exact, and if the target bandwidth is achieved for all paths, all of the offered traffic is carried.

2. For each link i

$$\sum_{P \in \mathcal{P}^{(i)}} B_P \leq b_i \tag{2}$$

   so that no link has an offered (target) load greater than its capacity.

We next choose a definition of optimality. Let

$$f_i = \sum_{P \in \mathcal{P}^{(i)}} B_P$$

denote the *target flow* on link i and let f_i/b_i denote the *target utilization* of link i. Let $s_i = b_i - f_i$ denote the *target slack* on link i. Constraint (2) implies that all slacks must be non-negative.

Let $F_i(f_i)$ denote a penalty function for link i when the link carries a flow f_i. The *LSP design problem* is specified in terms of the following constrained non-linear optimization problem: Find a set of feasible target bandwidths $\boldsymbol{B}_{\text{opt}}$ that minimizes the objective function

$$F(\boldsymbol{B}) = \sum_i F_i(f_i) \tag{3}$$

subject to the constraints (1) and (2) where the sum in equation (3) is over links i with $b_i > 0$. $\boldsymbol{B}_{\text{opt}}$ is said to provide an optimal solution to Eq. (3). Note that the optimal link flows f_i are almost certainly unique although the optimal bandwidths $\boldsymbol{B}$ are usually not: this matter is discussed in Appendix A.

2.3. The Penalty Function

The link penalty functions $F_i(x)$ used in the LSP design problem have at least three roles. First, they must to a reasonable degree represent an intuition of what constitutes a "good" load balancing scheme. Second, they must be an efficient way of managing constraints, in particular the constraint that no link carries a load larger than, or even close to, its bandwidth. Third, the penalty functions must make it possible to efficiently find an optimal solution to the LSP design problem.

The flow deviation algorithm requires that the penalty functions $F_i(x)$ be increasing and convex on $[0, b_i)$ with $\lim_{x \uparrow b_i} F_i(x) = +\infty$. The latter requirement necessitates a minor change to the definition of feasibility: a solution is said to be feasible if $f_i < b_i$ (strict inequality) on all links i. It is also convenient to make a slightly stronger demand on the functions $F_i(x)$ and require that each $F_i(x)$ be "strongly monotone" on the interval $[0, b_i)$ so that the penalty functions are non-negative on $[0, b_i)$ and their derivatives with respect to flow are positive on $(0, b_i)$.

Previous studies of the flow deviation algorithm[2–4] used

$$F_i(x) = M_i \frac{x}{b_i - x} \tag{4}$$

as a link penalty function. If we assume that the offered load to each link i is a Poisson process of packet arrivals and that packets have independent, identically distributed sizes with exponential distribution and average M_i, and that there is an infinite buffer, and that the resulting utilization of the link is x/b_i, then the penalty function (4) is the product of the flow x and the average delay (waiting and service both included, but propagation delay excluded). With the M/M/1 assumptions above, the sum of the link penalty functions is a measure of the average total network delay.

However, in the modern Internet with TCP, and RED and all its variations, it is possible to have very highly utilized links (utilization practically one) and still low delay and low loss in the buffer: all delay is moved to the edge of the network. The same holds for example for ATM with ABR, in particular the ER version of ABR. Equation (4) is probably no longer a suitable penalty function. Given these concerns, we present a link penalty function with properties which make it suitable for use in an objective function whose minimization will yield routes and bandwidths that correspond closely to the optimal operation of a modern internet. Our choice of penalty function is

$$F_i(x) = c_i x + \eta \sigma_i \left(\frac{\sigma_i}{b_i - x} \right)^{\nu} \tag{5}$$

where link i has a bandwidth $b_i \geq 0$, a weight factor $\sigma_i > 0$ with $\eta > 0$, $\nu > 1$ and $F_i(x) = \infty$ if $x \geq b_i$. The factor c_i is explained below. The function (5) is strongly monotone and the first derivative of the penalty function is

$$F_i'(x) = \frac{d}{dx} F_i(x) = c_i + \eta \nu \left(\frac{\sigma_i}{b_i - x} \right)^{\nu+1}. \tag{6}$$

Let $\tau_i \geq 0$ denote the propagation delay on link i. Set $F_i'(0) = \tau_i$. Then $c_i = \tau_i - \eta \nu \left(\sigma_i / b_i\right)^{\nu+1}$. The properties of the link penalty function (5) under light and heavy load are discussed in the following section.

2.4. Behavior under Light and Heavy Load

With reference to the link penalty function (5) we choose η positive but small so that if a feasible solution exists for which all flows f_i are small and all link utilizations f_i/b_i are low – in which case the system is said to be uniformly lightly loaded – then the penalty function (5) will yield routes that are in agreement with OSPF routing where the propagation delays are the OSPF metrics of the links.

If the system is not uniformly lightly loaded then the penalty function enforces a distance from the barrier b_i. The parameter η determines when the barrier b_i begins to dominate the initial linear behaviour of the penalty function. A larger value of η causes the penalty function to rise earlier when the flow approaches the barrier. The parameter ν determines the behaviour of the penalty function as it approaches the barrier b_i. A larger value of ν makes the penalty function steeper when the flow approaches the barrier.

Equation (6) shows that if ν is large and for some link i the entity

$$\frac{s_i}{\sigma_i} = \frac{b_i - f_i}{\sigma_i}$$

becomes both small in the absolute sense, and also becomes the smallest among all the links, then the dominating objective of the NLP becomes to increase that entity. We call s_i/σ_i the *weighted slack* of link i. Thus if ν is large then the NLP maximizes the minimal weighted slack, at least as long as that minimal weighted slack is small. The magnitude of the minimal weighted slack depends on ν. Even better: as long as ν is sufficiently large, the NLP attempts to perform a "lexicographic maximization" of all small weighted slacks: first it maximizes the smallest weighted slack, then the next smallest, and so on.

The choice of σ_i is of interest. For example, if we choose $\sigma_i = b_i$ then we minimize the maximal utilization, as long as that maximal utilization is large. An interesting situation also arises when all b_i (insofar positive) are large. In that case we can choose for σ_i an estimate of the standard deviation of the instantaneous offered load to link i in the situation where the target load is somewhat close to b_i. In that case the weighted slack is the "distance" from the

target flow f_i to the bandwidth b_i, measured in units of standard deviations. Assuming a Central Limit Theorem, and assuming that the distance as defined above is at least several standard deviations, then by maximizing the minimal weighted slack we are also essentially minimizing the maximal probability that the offered load to a link is larger than its bandwidth.

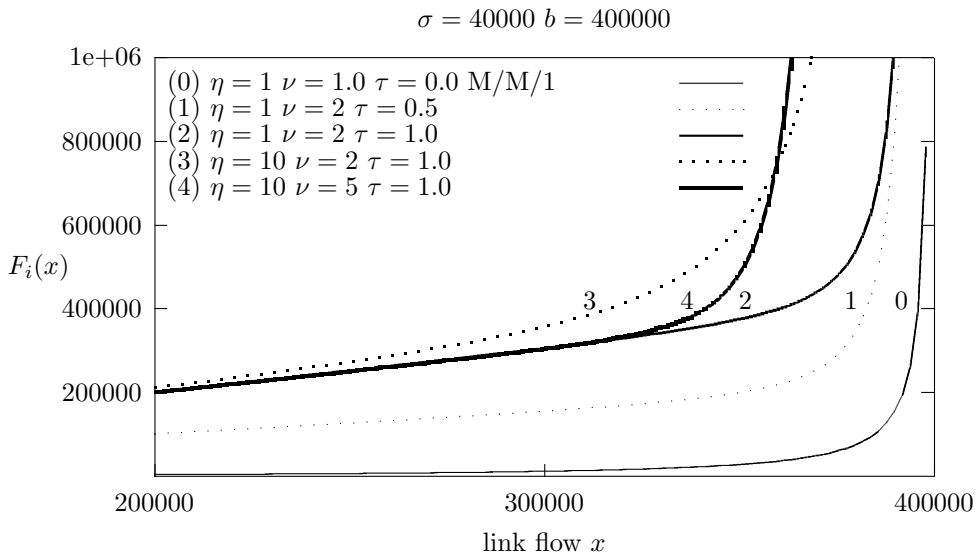

Figure 1. Examples of the penalty function

Figure 1 plots the penalty function (5) of a link i as a function of the link flow x. The link bandwidth $b_i = 400,000$ and the weight $\sigma_i = b_i/10 = 40,000$. These values are related to the parameters of a 50-node network model[8] where the average link capacity is 190,689 ±81,026 and the average flow carried on a link is 95,265 ±48,414. This model is analyzed in Sect. 4.

With reference to Fig. 1 plot (0) shows the M/M/1 penalty function using related parameters. Plots (1) through (4) are for the penalty function (5). Plot (1) shows the effect of $\tau_i = 0.5$, $\eta = 1$ and $\nu = 2$. Plot (2) shows the effect of increasing τ_i from 0.5 to 1.0. The parameter η determines when the barrier b_i begins to dominate the initial linear behaviour of the penalty function. Plot (3) shows that the penalty function begins to rise towards the barrier earlier when η is increased from 1 to 10. The parameter ν determines the behaviour of the penalty function as it approaches the barrier b_i: increasing ν increases the steepness of the rise. Plot (4) shows the effect of increasing ν from 2 to 5.

3. THE FLOW DEVIATION ALGORITHM

This section presents an implementation of the flow deviation algorithm[2–4] which minimizes a convex objective function and thus converges to a global optimum.

The operation of the flow deviation algorithm is simple. The algorithm executes in a loop where each iteration of the loop implements one step of the algorithm. During each step the algorithm computes the current set of shortest (least cost) paths from all sources to all destinations. An optimal amount of flow is diverted from the current set of LSPs to the shortest paths. Those shortest paths that are not already in the LSP set are added to the LSP set, the link costs are updated (the link costs have changed because the link flows have changed) and the next step of the algorithm is executed. The loop continues until flow re-distribution achieves no further reduction in the objective function. A small worked example of the operation of the flow deviation algorithm can be found in.[3]

3.1. The Algorithm

In the MPLS context the flow deviation algorithm incrementally improves the set $\mathcal{P}$ of LSPs and improves the distribution of traffic over multiple paths in $\mathcal{P}$ from the same source to the same destination. Improving $\mathcal{P}$ mainly consists of adding paths that have, or are likely to have, lower cost than the existing paths from the same source to

the same destination. Improving $\mathcal{P}$ may involve discarding paths P that are known not to have positive B_P in any optimal solution, or are not likely to have such a positive flow. Discarding non-promising paths is not necessary for convergence but significantly decreases the computational effort.

The algorithm executes in a loop. Each iteration of the loop implements one step which is identified by a step index k.

1. Initialize: Set $k = 0$. For each link i set the link flow $f_i = 0$. Compute the least cost path $P = P_e(m,n)$ connecting each node pair (m,n). Set the target bandwidth $B_P = d_{(m,n)}$. If necessary call step (6) to enforce a feasible solution. Initialize the path set $\mathcal{P} = \cup_{(m,n)} P_e(m,n)$.
2. For each link i compute the link cost $C_i^L = F_i'(f_i)$. For each path P compute the route cost $C_P^R = \sum_{i \in \mathcal{L}_P} C_i^L$.
3. Compute a feasible direction $\boldsymbol{\Delta} = (\Delta_P)_{P \in \mathcal{P}}$ and an improved path set $\mathcal{P}$. The calculation of an improved path set and a feasible direction is discussed in Sect. 3.2.
4. Convergence test: If no feasible direction can be found then optimality has been achieved and the algorithm halts. This stopping rule is theoretically correct but of no practical value. A practical stopping rule is discussed in Sect. 3.4.
5. Compute improved path flows $\boldsymbol{B}$: Compute a value of x such that $B_P := B_P + x\Delta_P$ yields a value $F(\boldsymbol{B})$ of the objective function which is a strict improvement over the value of the objective function computed in the previous step, and in the direction $\boldsymbol{\Delta}$ is optimal. This computation is called the "line search" for x. The calculation of x is discussed in Sect. 3.3. In qualitative terms: a very small positive x value always gives an improvement. We increase x either until the objective function stops decreasing, or until a path flow B_P goes to zero in which case the path P leaves the set $\mathcal{P}$.
6. Enforce a feasible solution: If the target bandwidths $\boldsymbol{B}$ are not feasible then for each link i set $b_i := \alpha b_i$ where $\alpha = \max_i(1.05 f_i / b_i)$. The solution $\boldsymbol{B}$ is now feasible. *
7. Compute improved link flows: For each link i compute $f_i := f_i + x\delta_i$ where $\delta_i := \sum_{P \in \mathcal{P}^{(i)}} \Delta_P$.
8. Loop statement: $k := k + 1$ and go to step 2.

3.2. Choosing a Feasible Direction

A feasible direction is a map $\boldsymbol{\Delta} = (\Delta_P)_{P \in \mathcal{P}}$ with the following properties:

- the traffic demand $d_{(m,n)}$ offered to each node pair (m,n) is constant therefore $\sum_{P \in \mathcal{P}_{(m,n)}} \Delta_P = 0$,
- an empty path cannot have its bandwidth allocation lowered so that if $B_P = 0$ then $\Delta_P \geq 0$,
- a feasible direction will lower the network cost so that $\sum_{P \in \mathcal{P}} \Delta_P C_P^R < 0$.

We present two methods for computing a feasible direction. The first method, the so-called global method, may add paths to the set $\mathcal{P}$. The second method, the so-called local method, does not add paths to the set $\mathcal{P}$: in fact it is likely to remove paths from $\mathcal{P}$.

3.2.1. The global method

Given a feasible solution $\boldsymbol{B}$ and the current link costs C_i^L, compute the shortest path $P_e(m,n)$ connecting each source-destination pair (m,n). There may be several such paths in which case a tie-breaking mechanism is needed. This path may already be in the set of known paths $\mathcal{P}_{(m,n)}$ and have a positive flow $B_{P_e(m,n)} > 0$. If the path is not in $\mathcal{P}_{(m,n)}$ then it is added to $\mathcal{P}_{(m,n)}$. For each $P \in \mathcal{P}_{(m,n)}$ compute

$$\Delta_P = \begin{cases} -B_P & P \in \mathcal{P}_{(m,n)} \setminus P_e(m,n) \\ d_{(m,n)} - B_P & P = P_e(m,n). \end{cases}$$

*When the flow deviation algorithm terminates then $\alpha = 1$ else the solution $\boldsymbol{B}$ is not feasible.

3.2.2. The local method

Given a feasible solution $\boldsymbol{B}$ and the current link costs C_i^L and the route costs C_P^R, choose a subset $\mathcal{R}_{(m,n)}$ from $\mathcal{P}_{(m,n)}$ for each source-destination pair (m,n) as follows: all routes $P \in \mathcal{P}_{(m,n)}$ with $B_P > 0$ are in $\mathcal{R}_{(m,n)}$; optionally some or all routes $P \in \mathcal{P}_{(m,n)}$ that have the minimal value of C_P^R for all $P \in \mathcal{P}_{(m,n)}$ may be included, even those with $B_P = 0$; no other paths are included in $\mathcal{R}_{(m,n)}$.

Let $\mathcal{R} = \cup_{(m,n)}\mathcal{R}_{(m,n)}$ denote the set of *active* paths. $\mathcal{R}$ includes the paths P that have $B_P > 0$ and in addition $\mathcal{R}$ may contain some paths P that, because of their low current cost compared with other active paths from the same source to the same destination, are likely to be assigned a positive B_P.

Let $R_{(m,n)} = |\mathcal{R}_{(m,n)}|$ denote the number of active paths that connect node m to node n. Let

$$C_{(m,n)}^S = \sum_{P \in \mathcal{R}_{(m,n)}} C_P^R.$$

For each $P \in \mathcal{R}_{(m,n)}$ compute

$$\Delta_P = C_{(m,n)}^S - R_{(m,n)} C_P^R. \tag{7}$$

Thus if there are for example two routes from node m to node n then Eq. (7) will decrease the flow on the more costly route and increase the flow on the cheaper route (at the same rate), and the rates of change are proportional to the difference in route costs. The ideal situation would be that where all node-pairs (m,n) with two routes will reach their cross-over point where costs become equal at about the same time (for about the same value of x).

Reducing the size of the path set $\mathcal{P}$ substantially improves the performance of the flow deviation algorithm. The next section describes a method to quickly remove many paths from $\mathcal{P}$ that are unlikely to belong to the final optimal set of paths.

The local method: removing inferior paths

Given a feasible direction $\boldsymbol{\Delta}$, compute for each source-destination pair (m,n)

$$x_{max}^R(m,n) \quad = \quad \min_{P \in \mathcal{R}_{(m,n)}: \Delta_P < 0} \quad \frac{B_P}{|\Delta_P|} \tag{8}$$

where $x_{max}^R(m,n) = +\infty$ if $\Delta_P = 0$ for all $P \in \mathcal{R}_{(m,n)}$. In the line search, if x grows to $x_{max}^R(m,n)$ then the flows on one or more of the routes in $\mathcal{R}_{(m,n)}$ will decrease to zero, and that route would be expelled from $\mathcal{P}$. This mechanism with high probability expels at most one route per iteration. We have a mechanism to improve this:

Choose a parameter $\widehat{x} \geq 0$. For those source-destination pairs (m,n) with $x_{max}^R(m,n) < \widehat{x}$ we re-scale

$$\Delta_P := \Delta_P \frac{x_{max}^R(m,n)}{\widehat{x}} \tag{9}$$

for all $P \in \mathcal{R}_{(m,n)}$. Compute

$$x_{max}^R = \min_{(m,n)} x_{max}^R(m,n)$$

using the values of $x_{max}^R(m,n)$ computed in Eq. (8) with the re-scaled $\boldsymbol{\Delta}$ values as computed in Eq. (9). Now $x_{max}^R \geq \widehat{x}$ and if Eq. (9) is applied at least once then $x_{max}^R = \widehat{x}$. The result now is that no path in $\mathcal{R}$ loses all its flow until x increases to x_{max}^R, and for $x = x_{max}^R$ a potentially large number of paths all lose all their flow.

We proceed as follows: initialize $\widehat{x} = 0$ so no rescaling occurs after Eq. (7) has computed a feasible direction $\boldsymbol{\Delta}$. If equations (11) and (12) below determine $x = x_{max} = x_{max}^R$, (i.e. the optimal x is one that causes elimination of at least one route), calculate

$$\widehat{x} = 2x_{max}^R$$

for use in the next iteration of the flow deviation algorithm. Else (if the optimal value of x does not cause elimination of any route) we set $\widehat{x} = 0$ for the next iteration of the flow deviation algorithm. Once the correct set of paths has been found, $\widehat{x}$ is likely to remain at zero.

3.2.3. A mixed method

It is likely that in the optimal solution each source-destination pair (m, n) will be connected by a small number of paths $P \in \mathcal{P}_{(m,n)}$ and these paths will all have the same route costs. It is to be expected that after a while the shortest path algorithm will keep returning paths from that small set. Once the algorithm is in this situation the use of the local method seems preferable.

The local and global methods can be combined as follows. The flow deviation algorithm initially iterates using the global method until the shortest path algorithm finds no new paths.

The flow deviation then alternates between the local and the global methods as follows: an iteration of the global method is followed by k iterations of the local method ($k \geq 0$) and then another iteration of the global method. If that new iteration of the global method finds a new path, it is followed by zero iterations of the local method. Otherwise, it is followed by $k + 1$ iterations of the local method.

The algorithm thus alternates between the global and local methods until the stopping rule in Sect. 3.4 below is triggered.

3.3. The Line Search

In this section we compute a value of x which yields an improved solution

$$B_P(x) = B_P + x\Delta_P \tag{10}$$

for all $P \in \mathcal{P}$. Define

$$x^L_{max} = \min_{i:\delta_i>0} \quad \frac{s_i}{\delta_i}$$

where $x^L_{max} = +\infty$ if $\delta_i \leq 0$ for all i. If x grows to $x^L_{max} < \infty$ then the slack on one or more links will equal zero. Thus we have the constraint $x < x^L_{max}$. Next define

$$x^R_{max} = \min_{P:\Delta_P<0} \quad \frac{B_P}{|\Delta_P|}.$$

It is impossible that $\Delta_P \geq 0$ for all P. If x grows to x^R_{max} then the flows on one or more routes in $\mathcal{P}$ will decrease to zero. Thus we have the constraint $x \leq x^R_{max}$.

Set $x_{max} = \min(x^L_{max}, x^R_{max})$. With an abuse of notation, we wish to find a value of $x \in [0, x_{max}]$ which minimizes

$$F(x) = \sum_i F_i(f_i + x\delta_i).$$

Setting $y = f_i + x\delta_i$ and taking derivatives we obtain

$$F^{(k)}(x) = \left(\frac{d}{dx}\right)^{(k)} F(x) = \sum_i (\delta_i)^k \left(\frac{d}{dy}\right)^{(k)} F_i(y).$$

Since the functions $F_i(\cdot)$ are strongly monotone, even derivatives of $F(x)$ are positive and odd derivatives of $F(x)$ are strictly increasing. If

$$x^R_{max} < x^L_{max} \tag{11}$$

then $x_{max} = x^R_{max}$. In that case, compute

$$F'(x_{max}) = F^{(1)}(x_{max}) \tag{12}$$

If Eq. (11) holds and $F'(x_{max}) \leq 0$ then x_{max} is the optimal value for x. In this case, in updating the feasible solution, one or more routes have their flow reduced to zero and these routes may be removed from $\mathcal{P}$.

If Eq. (11) does not hold, or if it holds but $F'(x_{max}) > 0$ then we need to find the value of $x \in [0, x_{max})$ where $F'(x) = 0$. Because the even derivatives of $F(x)$ are positive we can use the Newton-Raphson method[9] to find the value of x.

3.4. The Stopping Rule

The algorithm requires a stopping rule to determine the iteration k when the algorithm has converged. We can stop when either $|F'(x_{k+1}) - F'(x_k)|$ or $|x_{k+1} - x_k|$ has been close to zero for some time in which case further iterations will yield no improvement in the solution. We can use a combination of these two criteria. Because even derivatives of $F(\cdot)$ are positive, the sequence (x_k) will become monotone decreasing or increasing. It may be safe not to stop until the sequence has been monotone for some time *and* one or both of the other conditions above is satisfied.

3.5. Comparing the Local and Global Methods

The global method. The global method computes the shortest paths between all source-destination pairs each time a new feasible direction is calculated. The shortest path calculation has complexity $\mathrm{O}(N^3)$ where N is the number of nodes in the network. After each shortest path calculation the global method needs to check if the shortest paths are already in the set $\mathcal{P}$.

The choice of direction can lead to convergence problems. For many source-destination pairs there are, in the optimal solution, several paths with equal costs and each path carries a significant flow. When close to the optimal solution, one of these paths will have the least cost. The global method will move flow from the slightly more costly path to the slightly less costly paths. With high likelihood, in the next iteration the previous slightly more costly path has become the slightly less costly, and the direction of the transfer of flow is reversed: the algorithm oscillates.

The local method. The local method for choosing a feasible direction has computational complexity $\mathrm{O}(R_{(m,n)})$ per source-destination pair (m, n).

Paths which are likely not to carry a flow in an optimal solution are removed from the set of active flows by driving their flows to zero. If we had not used the refinement introduced in Sect. 3.2.2 then with near certainty the local method will remove one path per iteration or worse: not every iteration need eliminate one such flow. If there were a large number of such flows this could, in the absence of the above refinement, cause a major slowdown in the algorithm.

3.6. Implementation Issues

Each time the global method is invoked it calculates the shortest paths connecting all source-destination pairs and checks whether the current set of shortest paths is already in $\mathcal{P}$. These calculations are computationally expensive. The shortest paths are computed using Floyd's algorithm – see[10] and the references therein for a discussion of the relative merits of several well-known shortest path algorithms. Each path P is stored in a table which is accessed via a hash index computed over the link set $\mathcal{L}_P$.

4. EXPERIMENTAL RESULTS

This section presents several numerical studies of MPLS path selection in models of 50- and 100-node networks. A description of the models with their link capacities and offered traffics can be found at the URL http://www.cs.sun.ac.za/projects/COE/models.zip.

In the remainder of this section, the length of a path denotes the hopcount of a path. The *normalized length* of a path is the length of that path minus the length of the shortest path connecting the source-destination pair of that path.

A 100-node Network

The 100-node network has 244 uni-directional links and carries 1 traffic class. The links are capacitated with 6,515,881 units of bandwidth. A total of 250,000 units of flow are offered to the 9,900 source-destination pairs. The parameters of the penalty function (5) are $\eta = 1$, $\nu = 1$, $\tau_i = 1$ and $\sigma_i = b_i/10$.

The flow deviation algorithm was run twice using the global and the mixed methods. The global method requires some 20 seconds of CPU time on a Pentium III 700 MHz processor to solve the LSP design problem for the 100 node model; the mixed method requires some 10 seconds. Tables 1 and 2 confirm that the LSP sets computed by the two methods are nearly equivalent.

ρ	global	mixed
0.0 – 0.1	0	0
0.1 – 0.2	68	68
0.2 – 0.3	84	82
0.3 – 0.4	48	50
0.4 – 0.5	12	12
0.5 – 0.6	10	6
0.6 – 0.7	8	12
0.7 – 0.8	4	4
0.8 – 0.9	2	0
0.9 – 1.0	8	10

Table 1. 100-node network: distribution of link utilizations ρ

The global method finds an optimal LSP set containing 12,250 routes. The average normalized route length is 0.38 and the average LSP bandwidth is 20.1. The mixed method finds an optimal LSP set containing 12,051 routes. The average normalized route length is 0.37 and the average LSP bandwidth is 20.8. The LSP sets have several attractive features. The LSPs overwhelmingly coincide with the shortest routes connecting the source-destination pairs. Most source-destination pairs are connected by one or two LSPs, and nearly all of the flow is assigned to the shortest LSPs. Some 96% of the bandwidth is assigned to the shortest and the shortest-but-one LSPs.

Table 1 shows that the global and mixed methods yield nearly the same link utilization distributions.

A source-destination pair is said to have an n-path connection or a *path multiplicity* of n if the pair is connected by n LSPs. Table 2 shows the bandwidth assigned to n-path connections where $n = 1, 2, \ldots, 6$. For example the second row of Tab. 2 shows that 1,996 source-destination pairs are connected by two routes: the 3,992 routes carry 30,645 units of flow which is 12.2% of the total flow carried by the network. Each of these routes carries on average 7.6 units of flow. Each two-path connection carries on average 15.3 units of flow. The global method yields an average path multiplicity of 1.24. The mixed method yields an average path multiplicity of 1.22.

path multiplicity	S-D pairs	S-D routes	flow	%	flow/ route	flow/ S-D
global method						
1	7732	7732	217927	87.1	28.1	28.1
2	1996	3992	30645	12.2	7.6	15.3
3	164	492	1352	0.5	2.7	8.2
4	6	24	52	0.0	2.1	8.7
5	2	10	7	0.0	0.7	3.8
mixed method						
1	7901	7901	220571	88.2	27.9	27.9
2	1863	3726	28356	11.3	7.6	15.2
3	120	360	990	0.4	2.7	8.2
4	16	64	83	0.0	1.3	5.2

Table 2. 100-node network: path multiplicity

A 50-node Network

The 50-node network[8] has 202 uni-directional links and carries 1 traffic class. The links are capacitated with 38,519,241 units of bandwidth. A total of 6,581,372 units of flow are offered to the 2,450 source-destination pairs.

The flow deviation algorithm using the mixed method was applied to compute optimal LSP sets for the 50-node model as the offered traffic is increased by a factor K where K is 1.0, 1.25 and 1.50. The parameters of the penalty function (5) are $\eta = 1$, $\nu = 2$, $\tau_i = 1$ and $\sigma_i = b_i/10$. About 1 second of Pentium III 700Mhz CPU time is needed to solve each model.

Let R denote the number of routes in an optimal set of LSPs and let H denote their average normalized length. Table 3 shows that the number of routes decreases as the traffic factor K increases. As the traffic increases the flow deviation algorithm requires more iterations to converge: the larger the number of iterations, the more opportunity the local method has to eliminate inferior paths. Nearly all the routes have a normalized length of zero which indicates that the link penalty functions are operating in their linear regions where their cost is equal to the constant propagation delay $\tau_i = 1$. Finally, Tab. 4 shows how the path multiplicity decreases as the offered load increases.

	traffic factor K		
	1.0	1.25	1.50
R	3854	3411	3177
H	0.0	0.0	0.0

Table 3. 50-node network: number of routes R, average normalized route length H

	traffic factor K					
	1.0		1.25		1.50	
path multiplicity	S-D pairs	flow %	S-D pairs	flow %	S-D pairs	flow %
1	1525	62.4	1767	72.1	1877	76.6
2	605	24.7	481	19.6	440	17.8
3	207	8.2	143	5.8	112	4.8
4	76	3.3	44	2.0	21	0.8
5	29	1.2	13	0.5		
6	7	0.2	2	0.1		
7	1	0.0				

Table 4. 50-node network: path multiplicity

5. CONCLUSION

This paper considers the problem of optimal path selection in MPLS networks. The problem is formulated as the minimization of a non-linear objective function which under light load simplifies to OSPF routing with link metrics equal to the link propagation delays, and under heavy load minimizes the probability that a transmission link has an instantaneous offered load larger than its bandwidth. We present an efficient algorithm based on the flow deviation method to find the optimal paths and to assign optimal bandwidths to these paths. The method is applied to find and capacitate optimal LSPs for two single service network models with 50 and 100 nodes respectively.

APPENDIX A. PATH SETS AND ROUTE DEGENERACY

The optimal solution $\boldsymbol{B}$ computed by the flow deviation algorithm is not unique. For example consider the network[1] presented in Fig. 2 where traffic is offered from nodes 1 and 2 to node 6: the traffic demands are $d_{(1,6)} = 0.5$ and $d_{(2,6)} = 1.5$. All links have capacity $b_i = 2$ and have the same propagation delay and the same weight factor. The optimal link flows are

$$\begin{array}{llllll} f_{(1,3)} & = & 0.5 & f_{(2,3)} & = & 1.5 \\ f_{(3,4)} & = & 1.0 & f_{(3,5)} & = & 1.0 \\ f_{(4,6)} & = & 1.0 & f_{(5,6)} & = & 1.0 \end{array}$$

Let f_P denote the flow on path P. We can assign any flow z where $0 \le x \le 0.5$ to path $(1, 3, 4, 6)$ whereupon the flows assigned to other routes are

$$\begin{array}{llllll} f_{(1,3,4,6)} & = & z & f_{(1,3,5,6)} & = & 0.5 - z \\ f_{(2,3,4,6)} & = & 1.0 - z & f_{(2,3,5,6)} & = & 0.5 + z. \end{array}$$

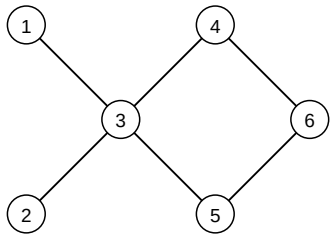

Figure 2. The "Fish" network

It is probably an advantage for a source-destination pair to have two paths rather than one path. Having four paths rather than three is probably a disadvantage. Operational requirements may prefer a particular value of z. Thus $z = 0$ and $z = 0.5$ will reduce the number of paths from four to three. The flow deviation algorithm yields $z = 0.25$ which assigns two paths from each of nodes 1 and 2 to node 6 with equal bandwidth. From the point of view of robustness under traffic forecast error, this may be the preferred solution.

Given the link flows, we need methods to compute not only a set of paths and a set of path flows consistent with the link flows, but we also need criteria to determine which set of paths and path flows are superior, and we need mechanisms to find optimal (according to those criteria) path sets and path flows.

REFERENCES

1. B. Davie, P. Doolan and Y. Rekhter, *Switching in IP Networks: IP Switching, Tag Switching and Related Technologies*, Morgan Kaufman Publishers Inc., 1998.
2. D. Bertsekas and R. Gallager, *Data Networks Second Edition*, Prentice-Hall International Inc., 1992.
3. A. Kershenbaum, *Telecommunication Design Algorithms*, McGraw-Hill, 1993.
4. L. Kleinrock, *Queueing System Vol. 2: Computer Applications*, John Wiley & Sons, New York, 1976.
5. S. Floyd and V. Jacobson, "Random early detection gateways for congestion avoidance", *IEEE/ACM Transactions on Networking*, 1993.
6. T.J. Ott, T.V. Lakshman and L.H. Wong, SRED: Stabilized RED, *Proceedings of IEEE INFOCOM'99*, pp. 1346 – 1355, 1999.
7. Traffic Engineering and QoS Methods for IP-, ATM- and TDM-Based Multiservice Networks. draft-ietf-tewg-qos-routing-01.txt. Obtainable from http://www.ietf.org/ietf/lid-abstracts.txt.
8. C. Villamizar, http://brookfield.ans.net/omp/random-test-cases.html.
9. W.H. Press, S.A. Teukolsky, W.T. Vetterling and B.P. Flannery, *Numerical Recipes in C Second Edition*, Cambridge University Press, 1992.
10. M.J. Atallah (ed), *Algorithms and Theory of Computation Handbook*, CRC Press, 1999.