IST2005
INTERNATIONAL SYMPOSIUM ON TELECOMMUNICATIONS
September 10-12, 2005 Shiraz, Iran

A New Off-line Algorithm for Batch End-to-End LSP Set up in MPLS Networks

Abdorasoul Ghasemi, Karim Faez
Department of Electrical Engineering
AmirKabir University of Technology, Hafez Avenue, Tehran, Iran, 15914
Emails: arghasemi@aut.ac.ir, kfaez@aut.ac.ir

Abstract

This paper presents a new algorithm based on dynamic programming to solve the problem of off-line LSP set up in MPLS networks. This algorithm is an offline algorithm that tries to map the LSPs onto a network graph such that a total cost function based on end-to-end delays and number of physical hops is optimized. The proposed algorithm consists of two steps. In the first step, we find k-shortest paths for each request and in the second step we try to assign one path to each request using dynamic programming algorithm. We approximate the original cost function with an additive one, which is a reasonable assumption for not very highly loaded links. We compare our results with the optimal solutions, which are resulted from exhaustive search. Results are presented as the percentage of times that the proposed algorithm can find optimal or suboptimal solution for a directed network graph and a set of LSP requests in different average link loads. The simulation result shows that the algorithm can find optimal or suboptimal solution in more than 94% of times.

Keywords: MPLS networks, LSP setup, Dynamic programming, k-shortest paths.

Introduction

Destination based forwarding, which is used traditionally in IP routers cannot take full advantage of multiple paths that exist in Internet service provider networks and prevents the networks to operate efficiently [1]. Development of Multi Protocol Label Switching (MPLS) that efficiently support explicit routing facilitates traffic engineering. Explicit routing allows a particular packet stream to follow a predetermined path rather than a path computed by hop-by-hop destination based routing such as Open Shortest Path First, OSPF, or Intermediate System-to-Intermediate System, IS-IS [1]. In MPLS, the packets are encapsulated, at ingress nodes, with labels that are then used to forward the packets along the Label Switched Paths (LSPs). The LSP can be thought of as virtual traffic trunk that carry flow aggregates by classifying the packets arriving at the edge or ingress routers of an MPLS network into "Forwarding Equivalence Classes" (FECs)[2].

Given the network graph, choosing the physical path to route a LSP request, is a central problem. In MPLS, paths can be calculated on-line as the demands arrive or it can be performed in the batch mode for a number of LSP requests arrived over a period based on long-term estimate of network traffic.

In on-line algorithms, requests arrive one-by-one and there is no a-priori knowledge about future requests. The objective of these algorithms is to accept as many requests as possible or minimize the request rejection. A well-known algorithm of this category is Minimum Interference Routing Algorithm, MIRA [2]. MIRA is an on-line algorithm that is based on the idea that a newly routed tunnel must follow a route that does not "interfere too much" with a route that may be critical to satisfy a future demand. The critical links are those links that if heavily loaded, would make it impossible to satisfy future demands between certain ingress-egress pairs.

Off-line calculation, which is usually used in network design process or traffic engineering try to design the network layout such that a cost function is globally optimized over the network graph [3, 4]. In other words the objective of off-line layout design is to make the most efficient use of the network, i.e., to minimize the resource usage for the LSPs that are being routed. With this objective, it is possible that after the routing has been done there may no available capacity between certain ingress-egress nodes. This lack of residual capacity will not be important if there is no future request. However, in MPLS network we expect to have on-line requests, therefore we can thought of off-line results as the initial operating point of the network.

Off-line layout design usually results into NP-complete problem and approximate solutions have been proposed before. In [5] a solution based on tabu search is proposed and in [3] a randomized algorithm is suggested.

In this paper we present a new two steps algorithm for this problem which with a low complexity can produce optimal or suboptimal solution. In the first step, we find k-shortest path for each request and in the second step we try to assign one path to each request using a dynamic programming algorithm. We approximate the original cost function with an additive one, which is a reasonable assumption for not very highly loaded links as happen in off-line layout design for MPLS networks. We compare our simulation results with optimal solutions, which are resulted from exhaustive search.

1. Problem definition

The network topology is modeled by a directed graph $G = (V, E)$, where V is the set of nodes (vertices) and E the set of links (edges). Each link e in E is associated with a number C_l which is denoting of the transmission capacity of that link. A LSP request is defined by a starting vertex s_i, an ending vertex d_i, a directed route from s_i to d_i, and a capacity, which also referred to as demand, C_i. It is assumed that the demand C_i is calculated from QOS requirements of the LSP request. The load of a link e is the summed capacity of LSPs traversing e. With these definitions, the problem statement is as follow: Given a list of source-destination pairs $\{(s_1, d_1),...,(s_M, d_M)\}$, each with a positive capacity demand C_i, the objective is to find a system $P = \{P_1, P_2,..., P_M\}$ of LSP paths such that the total end-to-end delay on network links, i.e., the equation (1) is minimized:

$$Cost = \sum_{all\ links\ (i,j)} \frac{F_{ij}}{C_{ij} - F_{ij}} \qquad (1)$$

P_i is the path connecting s_i to d_i for LSP request number i i.e., an end-to-end LSP. In equation (1) F_{ij} and C_{ij} are the flow and capacity of link (i, j) on the network graph.

Equation (1) is the total end-to-end delay assuming additive property for delay and is usually used as the cost function in the routing algorithms [6].

It is known that this problem is NP-complete and we should try to find approximate solutions [3].

2. K-shortest paths problem

The k-shortest paths problem is to find the k paths connecting a given source-destination pair in the given directed graph with minimum total length [7]. This problem is a generalized version of the well known shortest path problem. Given a directed graph, it was shown in [7] that the set of k shortest paths connecting a given pair of vertices can be found in time $O(|E| + |V| \log|V| + k)$, for each request, where $|E|, |V|$ are the number of edges and nodes. Therefore we can compute the k best candidate paths for each LSP request in a reasonable amount of time according to a certain criteria for path length. We should note that assigning the best path for each request is not possible because it may result into highly loaded or saturated links which is not acceptable and increases the cost function rapidly.

3. Proposed Algorithm

The proposed algorithm consists of two steps: finding k shortest (minimum hop count) paths for each request and assigning one path from k ones (which are calculated in step one) to each request.

There are two points to note about equation (1) First, the cost function will increase rapidly if the load of a link reaching the capacity of the link. Second, longer paths will have more effect on the cost function. The first factor addresses the load-balancing problem on the network and the second factor addresses the minimum hop routing scheme. Therefore, each strategy for designing LSP layout must consider both of these factors.

3.1. Finding K-shortest paths

Minimum hop algorithm is popular scheme that is used in online LSP setup. In this algorithm, the path from the ingress to the egress with the least number of feasible links is chosen. This can be achieved using the Dijkstra on the subgraph that is constructed from the network graph when the infeasible links i.e., links with lower capacity of the current request, are removed. Using minimum hop path decreases the resource consumption of the network. However, using minimum hop path for all requests makes some links saturated or congested and the cost function grows rapidly. Therefore we have to use longer paths for some requests according to the requests capacity and to minimize a given cost function to balance the network load on the various network links.

In the first step, considering the number of hop count as the length of a path, we find k shortest paths for each LSP request. Each of these paths can be a potential candidate for the route of the LSP request. Assigning one from k paths to each request is done in the second step of the algorithm. The larger the candidates i.e. k, the better solution and nearer to optimal, is found and also more computations are required. The Computational complexity of this work is given in section 2.

3.2. Path assignment

In the second step of the proposed algorithm we try to assign one path from k to each request. We use dynamic programming for these M sequential decisions, which has the computational complexity of $O(kM^2)$. Dynamic programming is a method to find the optimal solution in a class of sequential decisions with discrete time system and additive cost function over time. This method is based on the Bellman equation to find the optimal policy in a sequence of decisions. Using Dynamic programming algorithm an optimal policy can be constructed in piecemeal fashion, first constructing optimal policy for the tail sub problem involving the last stage, then extending the optimal policy for the tail sub problem involving the last two stages and continuing in this manner until an optimal policy for the entire problem is constructed.

In the second step of our algorithm, we consider the problem as a sequential decision problem with M, the number of LSP requests, stages that we should find an optimal policy in deciding one from k, the number of shortest paths calculated in stage one, options in each stage. Our problem is discrete in nature but the cost function that we are going to minimized is not additive over time. This is the problem that makes our algorithm to produce an approximate solution. In other word we are going to decide the LSP paths sequentially and hence the links flow. However the cost function of equation (1) is not additive with respect to links flow.

Therefore we approximate:

$$\frac{F_1+F_2}{C-(F_1+F_2)} \cong \frac{F_1}{C-F_1}+\frac{F2}{(C-F_1)-F_2}. \qquad (2)$$

This assumption is reasonable for not very highly loaded links, and is valid in off-line layout design as mentioned in introduction.

Briefly in the second step of our algorithm we use the approximated cost function and dynamic programming approach to find the optimal policy in assigning paths to requests. At the end of this step we assign one path to each LSP request.

The accuracy of this approximation depends on the links load because the smaller the term $F1+F2$ compared to C, the approximate equality is better provided in equation (2).

4. Numerical Results

Off line algorithm results, are usually compared to optimal solution which is resulted from the exhaustive search. In the exhaustive search all possible cases in assigning paths to requests are examined and the best assignment which yield the minimum cost is extracted.

To evaluate our algorithm we use the network graph of figure (1). The network graph consists of seven nodes and nine directed edges. There are three LSP requests i.e. $M=3$ that their sources and destinations are shown on the network graph.

We extracted four paths i.e. $k=4$ for each request considering hop count as the length of the path in the first step of the algorithm. For example the 4-shortest paths for LSP request between nodes 0 and 3 are:
$\{0-1-2-3, 0-4-2-3, 0-5-6-3, 0-4-5-6-3\}$.

In exhaustive search all possible 4^3 cases for path assignment is checked and the assignment with minimum cost is chosen as the optimal solution.

The larger the links capacities, the lighter the links load and the better the results are extracted from our algorithm. The reason is that in this case our additive assumption is more reasonable. Therefore we consider the average links load in our evaluations.

In each test, the LSP demands are randomly selected between $(0,1)$ and the network link capacities are randomly selected between $(0, X)$ where X is used to control average network link capacities or the average link loads. Each test is done hundred of times and the average performance of the algorithm to find optimal or suboptimal path is calculated and compared to optimal solution extracted from the exhaustive search.

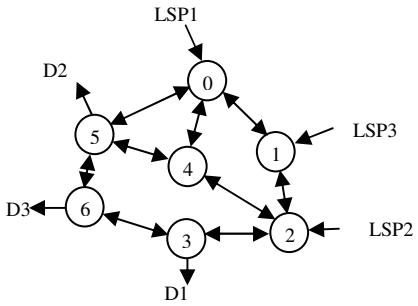

Fig. 1 Network graph for algorithm evaluation

Table 1 Performance Results of proposed algorithm

Average link load	% of times that reach optimal solution	% of times that reach sub optimal solution (% average deviation)	% of times that no solution was found
20 %	72	26.1 (14)	1.9
30%	71	23.4 (24)	5.6
40%	68	30 (10)	2

Table (1) summarized the results of the algorithm for three different average link loads.

Column two of this table shows the percentage of times that the proposed algorithm can find the optimal solution. Column three shows the percentage of times that the algorithm finds a sup optimal solution. The average deviation of this solution compare to optimal solution is given in parentheses. Last Column shows the percentage of times that the algorithm fails to find solution.

As the table shows, the lighter the links load the better is the performance of the proposed algorithm in percentage of times to find the optimal solution. Average deviation of the suboptimal solution is also an important factor that should be noticed.

An important advantage of our algorithm is its low computational complexity which makes it practical in large network graphs with hundreds of LSP requests where using exhaustive search is computationally impossible.

Conclusions

In this paper we have presented a new algorithm for off line LSP set up in MPLS networks. Off line layout design is usually used as an initial operating point for MPLS networks and in this phase links are not heavily loaded. Using this fact we proposed a new approximate algorithm based on dynamic programming to map LSP requests onto the network graph. In the first step of the algorithm we find k-shortest path for each request assuming hop count as the length of the path. In the second stage of the algorithm, approximating the cost function with an additive one, we try to assign one path to each request using dynamic programming method. Evaluations which are done on a directed network graph for different average links load, as summarized in table 1, show that the algorithm can find optimal or suboptimal solution in more than 94% of times. The main advantage of the proposed algorithm is its low complexity which makes it practical in real large network graphs.

References

[1] A. Elwalid, S. H. L. C. Jin, and I. Widjaja, "MATE: MPLS adaptive traffic engineering", *IEEE INFOCOM,* 2001.

[2] T. V. Lakshman, K. Kar, M. Kodialam, "Minimum interference routing of bandwidth guaranteed tunnels with mpls traffic engineering applications", *IEEE Journal on selected areas in communication,* Volume 18, Number 12, Dec. 2000

[3] I. Chlamtac, A. Farago, T. Zhang , "Optimizing the System of Virtual Paths", *IEEE/ACM Transactions on Networking,* Volume: 2, Issue: 6, Dec 1994

[4] H. Hsu, F. Yeang-Sung Lin, "Near-Optimal Constrained Routing in Virtual Circuite Networks", *IEEE INFOCOM* 2001.

[5] S. Beker, N. Puech, V. Friderikos, "A Tabu Search Heuristic for the Offline MPLS Reduced Complexity Layout Design Problem", *Lecture Notes in Computer Science*, Volume 3042 / 2004

[6] D. Bertsekas, R. Gallager, *Data Networks.* Englewood Cliffs, NJ: Prentice-Hall;2nd ed., 1992

[7] D. Eppstein, "Finding the *k* shortest paths", *SIAM J. Computing* 28(2):652-673, 1998