Building Alternate Multicasting Trees in MPLS Networks

Technical Report UTD/EE/3/2009

May 2009

Limin Tang, Shreejith Billenahalli, Wanjun Huang, Miguel Razo,
Arularasi Sivasankaran, Hars Vardhan, Marco Tacca, and Andrea Fumagalli
Open Networking Advanced Research (OpNeAR) Lab
Erik Jonsson School of Engineering and Computer Science
The University of Texas at Dallas, Richardson, TX, USA
andreaf@utdallas.edu
Paolo Monti
Next Generation Optical Network (NeGONet) Group
School of Information and Communication Technology, ICT-FMI
The Royal Institute of Technology, Kista, Sweden
pmonti@kth.se

Abstract

An algorithm for computing alternate multicast trees in packet transport networks is proposed in this paper. The algorithm efficiently computes multiple sub-optimal tree candidates for a given multicast service request. The algorithm builds on the widely used computation of K ordered loopless shortest paths and can be applied to any connected network topology. Simulation experiments obtained for a multiprotocol label switching (MPLS) network are presented to evaluate the effectiveness and performance of the algorithm.

Keywords: MPLS network, multicast, alternate tree

1. Introduction

In MPLS network routing, quite often the shortest path is not the favorable choice for a unicast label switched path (LSP). Having multiple alternate paths to choose from is a way to provide more options for the LSP route. The availability of multiple paths increases system flexibility and improves network optimization. One practical and simple approach to computing alternate paths for a given unicast LSP is to find the K shortest paths, i.e., a technique that has been used extensively to solve network optimization problems. Various algorithms are readily available to compute the K shortest paths from a source to a destination node [1], [2], [3], [4], [5], [6], [7]. However, for a multicast LSP request, previous work on computing alternate multicast trees is quite limited. A distributed algorithm to compute two nearly disjoint multicast trees in wireless ad hoc network is proposed by Wei and Zakhor [8]. However, the algorithm computes only two alternate trees, and it is specifically designed to account for node mobility.

The contribution of this paper is the design of a time efficient algorithm to compute multiple alternate trees for a given multicast LSP in MPLS network, assuming that every node in the network has unlimited multicast capabilities. Complexity of the algorithm is comparable to the complexity of the K shortest path algorithm. The paper is organized as follows. Section 2. describes the algorithm, derives its complexity, and proves its correctness. In Section 3., simulation results are discussed to evaluate the effectiveness of the proposed algorithm. The results are obtained by applying the algorithm to compute the multicast tree routes in MPLS network with the objective of minimizing the total cost of

the network equipment required in the network to support a given set of multicast LSPs. Section 4. contains some observations about future work on the subject.

2. Algorithm

This section contains the description of the algorithm to compute multiple alternate trees for a given multicast LSP request. The algorithm first computes multiple ranking loopless shortest paths from source to each of destination in the multicast tree. Multiple alternate trees are then computed based on these paths. The following notations are used:

- N: number of vertices in the network;
- M: number of edges in the network;
- s: source of the request;
- D: set of destinations of the request;
- d_i: the i^{th} destination of the request;
- n: number of destinations;
- P: set of ordered loopless paths from s to all $d_i \in D$;
- P_i: set of ordered loopless paths from s to d_i;
- p_{ij}: the jth shortest path from s to d_i;
- T: set of alternate multicast trees;
- K: minimum number of alternate trees to be computed;
- m: number of maximum hops from s to all $d_i \in D$; if no such constraint, $m = \infty$.

The algorithm is split into two steps or procedures.

Procedure 1 creates set P_i, i.e., it computes a set of K loopless shortest paths for every pair (s,d_i), $d_i \in D$.

Procedure 1:

For ($d_i \in D$)

  $P_i \leftarrow \emptyset$

  Compute K shortest paths from s to d_i

  For (jth shortest path)

    $p_{ij} \leftarrow$ the jth shortest path from s to d_i

    $P_i = P_i \cup p_{ij}$

  ENDFor

ENDFor

Procedure 2 makes use of the paths in set P_i to compute the trees to be added to T. Set T is set to be empty before running this procedure. The procedure works iteratively, adding one tree at a time. At each iteration, a new tree $G(V, E)$ — V is the set of vertices and E is the set of edges in the tree — is computed as follows. First, set V and E to be empty sets. Vertices and edges are increasingly added till a satisfactory tree is obtained. One at the time, all destinations $d_i \in D$ are considered as follows. Choose a path $p_{ij} \in P_i$, that was not used in any previous iteration. Add all the vertices and edges in the path to G. Then for each of the remaining destinations $d_{i'}$ choose one of the paths in set $P_{i'}$. Vertices and edges in this path are added to tree $G(V, E)$ starting from $d_{i'}$ and traveling backward along the path, until one edge that has a node that already belongs to tree $G(V, E)$ is added.

Then, move to the next destination node $d_{i'}$ The pseudocode for Procedure 2 is given next.

Procedure 2:

$k \leftarrow 0$

$T \leftarrow \emptyset$

`For` ($P_i \in P$)

  `For` ($p_{ij} \in P_i$)

    create a graph $G(V, E)$, $V \leftarrow \emptyset$, $E \leftarrow \emptyset$

    `For` vertex $v \in p_{ij}$

      $V = V \cup v$

    `EndFor`

    `For` edge $e \in p_{ij}$

      $E = E \cup e$

    `EndFor`

    `For` $d_{i'} \in D$ and $i' \neq i$

      $v \leftarrow d_{i'}$

      $e \leftarrow$last edge of $p_{i'1}$

      `While` ($v \notin V$)

        $V = V \cup v$

        $E = E \cup e$

        $v = v$'s upstream vertex on $p_{i'1}$

$e = e$'s upstream edge on $p_{i'1}$

EndWhile

EndFor

$t \leftarrow$the tree in $G(V, E)$

EndFor

If ($t \notin T$)

$acceptable \leftarrow$ True

If ($m < \infty$)

For ($d_i \in D$)

$h_i \leftarrow$number of hops from s to d_i in t

If ($h_i > m$)

$acceptable \leftarrow$ False

Break

EndIf

EndFor

EndIf

If ($acceptable =$ True)

$T = T \cup t$

EndIf

k++

EndIf

If ($k = K$)

```
      Break
    EndIf
  EndFor
```

2.1 Algorithm Complexity

This section evaluates the complexity of the proposed algorithm.

Procedure 1 computes K shortest paths from source to all destinations of the multicast traffic. Since computing K shortest paths for a pair of vertices has complexity $O(KN(M + N \log N))$ [6], the complexity of procedure 1 is $O(nKN(M + N \log N))$.

Procedure 2 has, at most, K iterations. Each iteration has three steps:

1. add vertices and edges of the j^{th} shortest path from s to d_i to the tree;
2. add vertices and edges of the shortest path from s to $d_{i'}(i' \neq i)$ to the tree;
3. if $m < \infty$, count number of hops from s to each d_i in t.

Since a shortest path can have at most M edges and at most $M + 1$ vertices, step 1 has complexity $O(M)$; similarly step 2 has complexity $O((n - 1)M)$; complexity of step 3 is $O(nM)$ since number of hops between any pair of vertices is at most M; so the complexity of procedure 2 is $K(O((n - 1)M) + O(M) + O(n))$ = $O(KnM)$.

Hence, the maximum complexity of the algorithm is $O(nKN(M+N \log N))$ + $O(KnM)$ = $O(nK(N+1)M+nKN \log N))$ = $O(nKN(M+N \log N))$, which means the proposed algorithm complexity is comparable to the complexity of computing K shortest paths from a given source node to each destination, i.e., procedure 1's complexity.

2.2 Proof of Correctness

In this section it is demonstrated that if multicast request has no maximum hop constraint($m = \infty$), the algorithm can build K distinct trees for the request if for at least one destination d_i, K shortest paths from s to d_i can be found.

For the proof of correctness for procedure 1, see [1] and [6]. The proof of correctness for procedure 2 is split into two parts. First, it is proven that $G(V, E)$ is a tree, then it is proven that for each $p_{ij} \in P_i$, a different tree is built. Without loss of generality, we assume that Procedure 2 begins with $i = 1$ and that there are K shortest paths from s to d_1.

Proof 1: $G(V, E)$ is a tree.

At the beginning, G only contains vertices and edges in p_{1j}, so it is loopless and is a tree with no branches. For destination $d_i(i \in \{2, \ldots, n\})$, edges and vertices of p_{i1} (shortest path from s to d_i) are added sequentially to G, beginning from the last edge and vertex (d_i), until the vertex is found in G. From these steps, we can see that:

- since p_{i1} is a loopless path and edge and vertex adding stops once the tree is reached, G is loopless;
- p_{i1}'s first vertex is s, so the tree will be eventually reached (since s is always in the tree).

Based on the observations above, we can see that step after step, G remains a tree.

Proof 2: For each $p_{1j} \in P_1$, a different tree is built.

Let T_j and $T_{j'}$ be the two trees built based on p_{1j} and $p_{1j'}$. First, we observe that all edges and vertices in $p_{i1}(i \in \{2, \ldots, n\})$ added to G are part of the shortest path tree(SPT); when $j = 1$, the built tree (T_1) is SPT. Then, we observe that for any two paths p_{1j} and

$p_{1j'}$, there are at least two edges $e(v_1, v_2) \in p_{1j}$ and $e'(v_1', v_2') \in p_{1j'}$ in the network that satisfy $v_1 \neq v_1'$ and $v_2 = v_2'$. Obviously, $e \notin p_{1j'}$ and $e' \notin p_{1j}$; also, e and e' cannot both be in SPT, otherwise we will have two shortest paths from s to v_2, which is not possible; so either e or e' is not in SPT. Without loss of generality, we assume e is not in SPT, then e cannot appear in any path of p_{i1}, combined with $e \notin p_{1j'}$, we have $e \notin T_{j'}$. We know $e \in p_{1j}$ hence $e \in T_j$, so T_j and T_j' must be two different trees.

1. and 2. prove that a tree built based on p_{1j} is distinct from a tree built based on $p_{1j'}$ when $j \neq j'$. Since $j \in \{1, \ldots, K\}$, at least K distinct trees can be built by the algorithm.

The proof above is based on the assumption that the multicast request has no constraint on the number of hops from source to any destination. However, if the request has such constraint, then the algorithm will not guarantee that K distinct alternate trees can be found. This is quite obvious since if such constraint exists, even shortest path between source and a destination may not be found; in an extreme case when $m = 1$, no multicast tree can be built unless all destinations are 1 hop away from the source, which is highly unlikely in reality.

3. Experiments

We designed two experiments to examine the effectiveness of the algorithm. In Experiment I, we mainly concerned the value of K's effect on the whole network optimization; in Experiment II, performance of the algorithm is evaluated when LSPs have constraint which does not allow hop count from source to destination to exceed certain number.

3.1 Experiment I

The following experiments are designed: three MPLS network topology are randomly generated, network 1 has 10 vertices and 60 unidirectional edges, network 2 has 20 vertices and 120 unidirectional edges and network 3 has 50 vertices and 300 unidirectional edges. For each edge in the network, it has maximum transmission capacity C. A number of multicast requests, with average number of destinations ranging from 2 to 8 and bandwidth request $C/100$, are randomly generated for each network.

Simulated Annealing (SA) algorithm is used in the experiment to find the optimal usage of bandwidth, which is to minimize the number of required links in the network. SA keeps looking for a better solution during a certain amount of time by creating an alternate solution each time, which is generated by letting each request randomly choose one from all available trees for multicast. The purpose of these experiments is to find out degree of optimization of the whole network when alternate trees are available for multicast LSPs.

Results of optimization are shown Fig. 1, 2 and 3. From these figures we can see clearly that in all three networks, total number of edges used by all requests is less when alternate trees are provided, optimization is significant especially when total number of requests is relatively low, which is reasonable, since when load of the whole network is low, there is more flexibility and it is more likely to get better optimization.

We can also see that increasing K usually brings much more gains when K is smaller than when K is large. This is natural since when the number of available trees is small, one more alternate tree can increase system flexibility a lot; but if the number of available trees

is large, improvement of one more alternate tree for optimization is relatively minimal and can even cause a reverse effect in some cases.

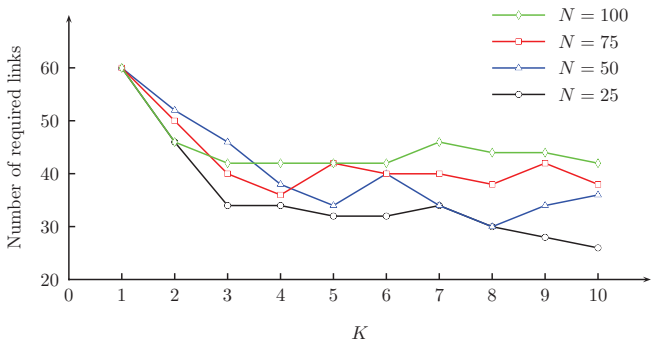

Figure 1

Effect of K on number of used edges for multiple multicast requests. The network has 10 vertices and 60 unidirectional edges, N is number of multicast requests.

3.2 Experiment II

Experiment II is similar to Experiment I, except that each request will have an extra constraint: hop count from source to each of the destinations cannot exceed certain value (m). Three cases are taken from Experiment I, which are network 1($|V| = 10, |E| = 60$) with 25 multicast requests, network 2($|V| = 20, |E| = 120$) with 50 multicast requests and network 3($|V| = 50, |E| = 300$) with 75 multicast requests. Results of optimization are shown in Fig. 4, 5 and 6 under different maximum hop count constraints. From Experiment I we can see that $K = 5$ usually provides fairly good optimization, so we set $K = 5$

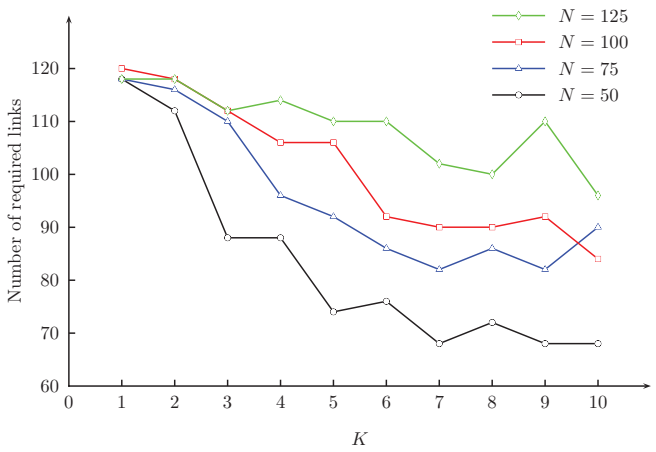

Figure 2

Effect of K on number of used edges for multiple multicast requests. The network has 20 vertices and 120 unidirectional edges, N is number of multicast requests.

in all cases in Experiment 2, i.e. each multicast request can have maximum 5 alternate trees built.

From these figures, we can see that even when multicast requests have maximum hop count constraint, having multiple candidate trees still provide reasonable network optimization, although not as good as without constraint, especially when the constraint is strict. In fact, when m is very small (e.g. 1 in network 1 and 2 in network 2), even SPT cannot satisfy the constraint, so there is no infeasible solution under the given constraint.

3.3 Experiment III

The following experiments are designed: three MPLS network topology are randomly generated, network 1 has 10 vertices and 80 unidirectional edges, network 2 has 20 vertices

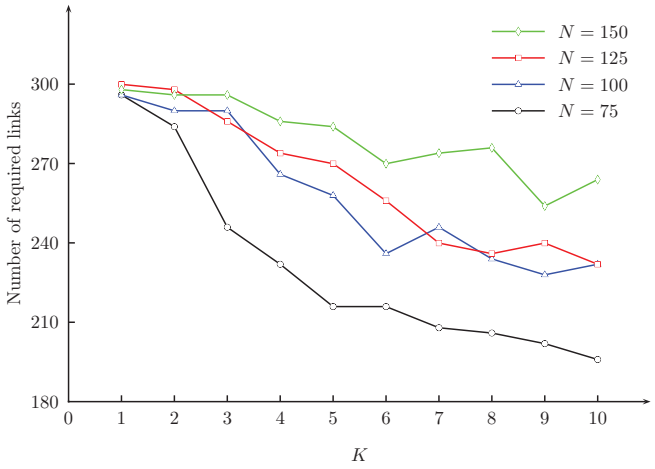

Figure 3

Effect of K on number of used edges for multiple multicast requests, the network has 50 vertices and 300 unidirectional edges, N is number of multicast requests.

and 160 unidirectional edges and network 3 has 50 vertices and 400 unidirectional edges. For each edge in the network, it has maximum transmission capacity C. A number of multicast requests, with average number of destinations ranging from 2 to 8 and bandwidth request $C/100$, are randomly generated for each network.

Simulated Annealing (SA) algorithm is used in the experiment to find the optimal usage of bandwidth, which is to minimize the number of required links in the network. SA keeps looking for a better solution during a certain amount of time by creating an alternate solution each time, which is generated by letting each request randomly choose one from all available trees for multicast. The purpose of these experiments is to find out degree of optimization of the whole network when alternate trees are available for multicast LSPs.

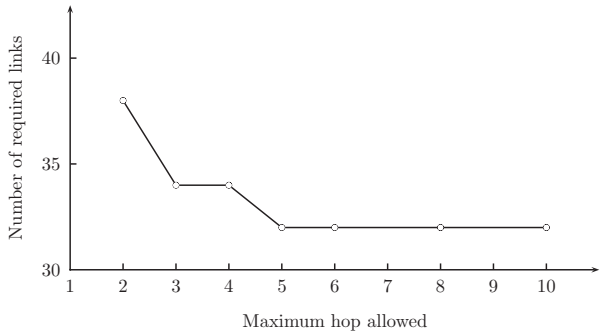

Figure 4

Effect of maximum hop count on number of used edges for multiple multicast requests. The network has 10 vertices and 60 unidirectional edges, number of multicast requests is 25 and $K = 5$.

Results of optimization are shown Fig. 7, 8 and 9. From these figures we can see clearly that in all three networks, total number of edges used by all requests is less when alternate trees are provided, optimization is significant especially when total number of requests is relatively low, which is reasonable, since when load of the whole network is low, there is more flexibility and it is more likely to get better optimization.

We can also see that increasing K usually brings much more gains when K is smaller than when K is large. This is natural since when the number of available trees is small, one more alternate tree can increase system flexibility a lot; but if the number of available trees is large, improvement of one more alternate tree for optimization is relatively minimal and can even cause a reverse effect in some cases.

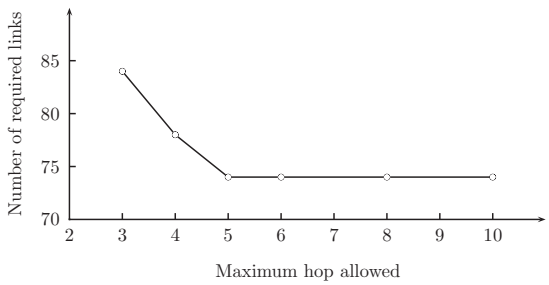

Figure 5

Effect of maximum hop count constraint on number of used edges for multiple multicast requests. The network has 20 vertices and 120 unidirectional edges, number of multicast requests is 50 and $K = 5$.

4. Conclusion

This paper presents an algorithm for computing alternate multicast tree candidates in a connected network topology. Both the algorithm and its complexity are based on the computation of K shortest paths — a widely used algorithm in networking. The purpose of computing multiple tree candidates for every individual multicast service request is to provide the network control plane with multiple options to choose from for every service, while optimizing some global cost function, e.g., bandwidth utilization, percentage of blocking. The effectiveness of the algorithm in computing alternate tree candidates was tested using a MPLS network example.

The considered computation of both the K shortest paths and alternate tree candidates accounts for the maximum hop count constraint only. However, multicast in MPLS networks may be subject to other types of constraint, e.g., limited multicast functionality at

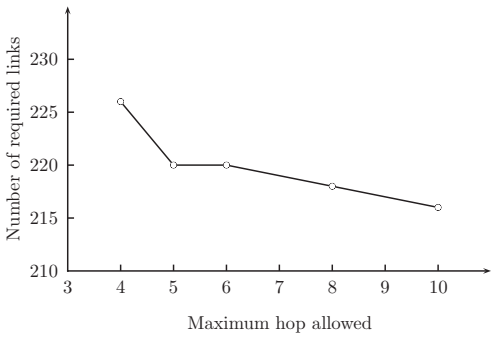

Figure 6

Effect of maximum hop count on number of used edges for multiple multicast requests. The network has 50 vertices and 300 unidirectional edges, number of multicast requests is 75 and $K = 5$.

the router. Additional research work is required to extend the proposed algorithm to compute multiple alternate tree candidates, while accounting for these additional constraints.

References

[1] J. Y. Yen, "Finding the k shortest loopless paths in a network," *management Science*, vol. 17, p. 712, 1971.

[2] D. Eppstein, "Finding the k shortest paths," *SIAM J. Comput.*, vol. 28, no. 2, pp. 652–673, 1999.

[3] A. Perko, "Implementation of algorithms for k shortest loopless paths," *Networks*, vol. 16, no. 2, p. 88, 1986.

[4] V. M. Jiménez and A. Marzal, "Computing the k shortest paths: A new algorithm and an experimental comparison," in *WAE '99: Proceedings of the 3rd International Workshop on Algorithm Engineering*. London, UK: Springer-Verlag, 1999, pp. 15–29.

[5] E. Hadjiconstantinou and N. Christofides, "An efficient implementation of an algorithm for finding k shortest simple paths," *Networks*, vol. 34, no. 2, p. 88, 1999.

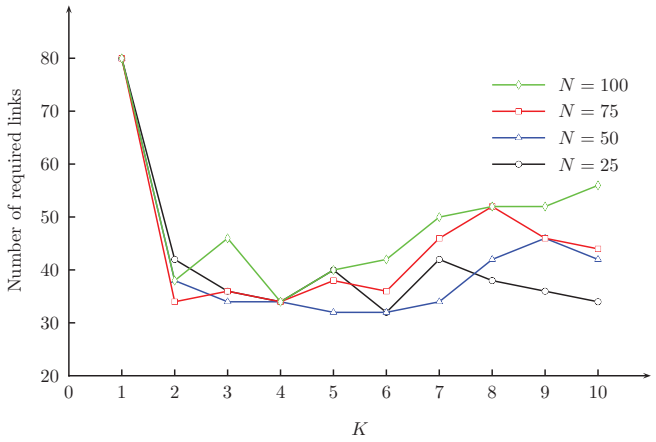

Figure 7

Effect of K on number of used edges for multiple multicast requests. The network has 10 vertices and 80 unidirectional edges, N is number of multicast requests.

[6] E. Q. Martins and M. M. Pascoal, "A new implementation of yen's ranking loopless paths algorithm," *Quarterly Journal of the Belgian, French and Italian Operations Research Societies*, vol. 1, p. 121, 2003.

[7] J. Hershberger, M. Maxel, and S. Suri, "Finding the k shortest simple paths: A new algorithm and its implementation," *ACM Trans. Algorithms*, vol. 3, no. 4, p. 45, 2007.

[8] W. Wei and A. Zakhor, "Multiple tree video multicast over wireless ad hoc networks," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 17, no. 1, pp. 2–15, Jan. 2007.

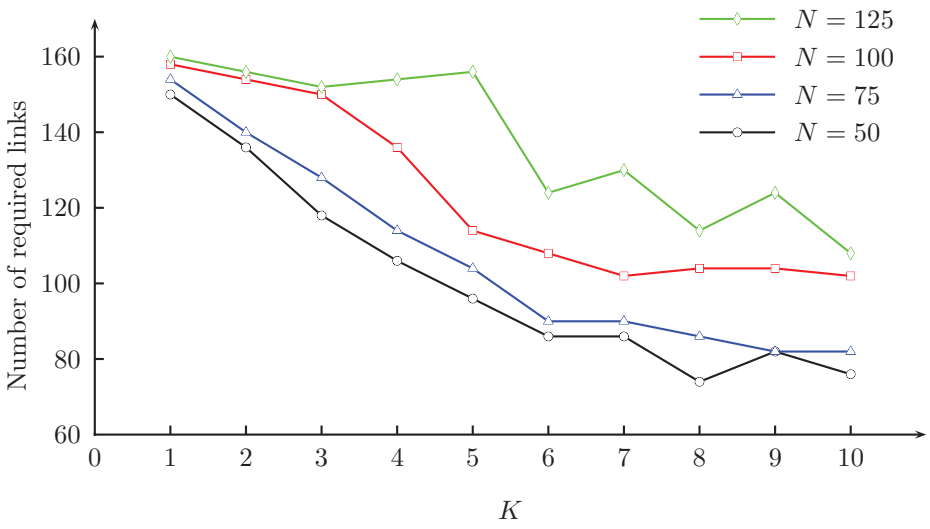

Figure 8

Effect of K on number of used edges for multiple multicast requests. The network has 20 vertices and 160 unidirectional edges, N is number of multicast requests.

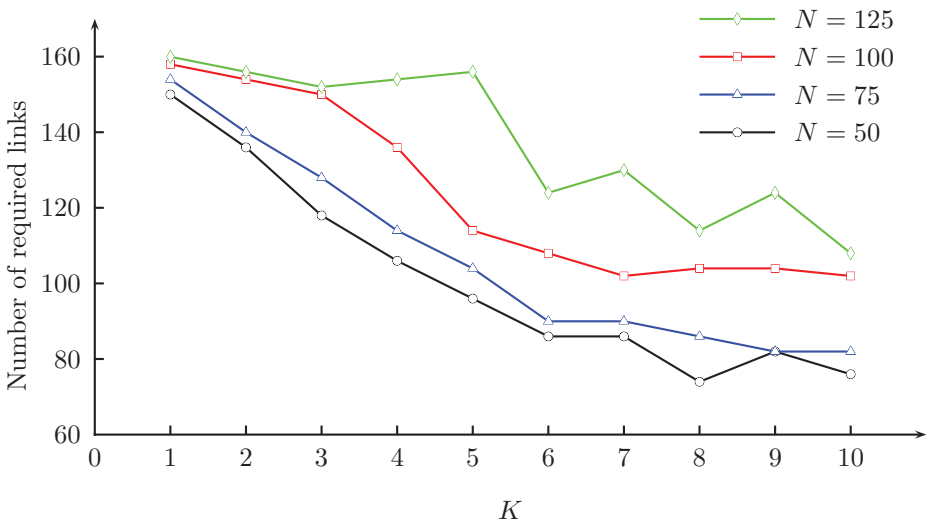

Figure 9

Effect of K on number of used edges for multiple multicast requests, the network has 50 vertices and 400 unidirectional edges, N is number of multicast requests.