A Delay-Constrained Shortest Path Algorithm for Multicast Routing in Multimedia Applications

Mohamed F. Mokbel
Department of Computer Science
Purdue University
West Lafayette, Indiana 47907
e-mail: mokbel@cs.purdue.edu

Wafaa A. El-Haweet
Department of Computer Science
and Automatic Control,
Alexandria University, Egypt.
e-mail: w_elhaweet@yahoo.com

M. Nazih El-Derini
Department of Computer Science
and Automatic Control,
Alexandria University, Egypt.
e-mail: nazih-s@safwa.com

Abstract

A new heuristic algorithm is proposed for constructing multicast tree for multimedia and real-time applications. The tree is used to concurrently transmit packets from source to multiple destinations such that exactly one copy of any packet traverses the links of the multicast tree. Since multimedia applications require some Quality of Service, QoS, a multicast tree is needed to satisfy two main goals, the minimum path cost from source to each destination (Shortest Path Tree) and a certain end-to-end delay constraint from source to each destination. This problem is known to be NP-Complete. The proposed heuristic algorithm solves this problem in polynomial time and gives near optimal tree. We first mention some related work in this area then we formalize the problem and introduce the new algorithm with its pseudo code and the proof of its complexity and its correctness by showing that it always finds a feasible tree if one exists. Other heuristic algorithms are examined and compared with the proposed algorithm via simulation.

Keywords: *Multicast Routing, Multimedia Applications, Real-Time Networks, Delay Constrained Tree.*

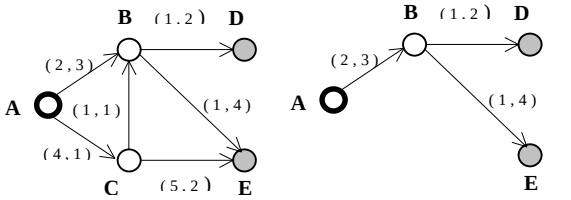
1. Introduction

Handling group communication is a key requirement for numerous applications that have one source sends the same information concurrently to multiple destinations. Multicast routing refers to the construction of a tree rooted at the source and spanning all destinations. Generally, there are two types of such a tree, the Steiner tree and the shortest path tree. Steiner tree or group-shared tree tends to minimize the total cost of the resulting tree, this is an NP-Complete problem, number of heuristics to this problem can be found in [1,2]. Shortest path tree or source-based trees tends to minimize the cost of each path from source to any destination, this can be achieved in polynomial time by using one of the two famous algorithms of Bellman [3] and Dijkstra [4] and pruning the undesired links. Recently, with the rapid evolution of multimedia and real-time applications like audio/video conferencing, interactive distributed games and real-time remote control system, certain QoS need to be guaranteed in the resulted tree. One such QoS, and the most important one, is the end-to-end delay between source and each destination, where the information must be sent within a certain delay constraint Δ. By adding this constraint to the original problem of multicast routing, the problem is reformulated and the multicast tree should be either delay-constrained Steiner tree, or delay-constrained shortest path tree. Delay constrained Steiner tree is an NP-Complete problem [5], several heuristics are introduced for this problem [5,6,7,8,9,10] each trying to get near optimal tree cost, without regarding to the cost of each individual path for each destination. Delay-constrained shortest path tree is also an NP-Complete problem [12]. An optimal algorithm for this problem is presented at [5], but its execution time is exponential and used only for comparison with other algorithms. Heuristic for this problem is presented in [11], which tries to get a near optimal tree from the point of view of each destination without regarding the total cost of the tree. An exhaustive comparison between the previous heuristics for the two problems can be found in [12,13]. In this paper we investigate the problem of delay constrained shortest path tree since it is appropriate in some applications like Video on Demand (VoD), where the multicast group has a frequent change, and every user wants to get his information in the lowest possible cost for him without regarding the total cost of the routing tree. Also shortest path tree always gives average cost per destination less than Steiner tree. We present a new heuristic algorithm that finds the required tree in polynomial time. The paper is arranged as follows: the problem is defined in section 2, the new algorithm is proposed in section 3 with the proof of its complexity and its correctness. Performance analysis by comparing the new algorithm with other previous algorithms is introduced in section 4. Section 5 contains the conclusions and the main contribution of this paper.

2. Problem Definition

The communication network is modeled as a directed, simple, connected weighted graph *G=(V,E)*, where *V* is the set of nodes and *E* is the set of directed links. Each link *e* in *E* connects two nodes *u, v* in *V* and is represented as *e(u,v)*. Two non-negative real value functions are associated with each link, the cost function *Cost(u,v)* represents the utilization of the link and the delay function *Delay(u,v)* represents the delay that the packet experiences through passing that link including switching, queuing, transmission and propagation delays. Links are asymmetrical, *Cost(u,v)* and *Delay(u,v)* do not necessarily equal *Cost(v,u)* and *Delay(v,u)*. A sequence of links that connects two nodes *u, v* are represented by a *Path(u,v)* with *Cost(Path(u,v))* which is equal to the sum of the costs of all its links and *Delay(Path(u,v))* which is equal to the sum of the delays of all its links.

Multicast group $M \subseteq V$ is a set of nodes that receives packets from source $S \in M$. The least cost tree (LC) is a tree originating at the source *S* and spanning all members of *M* with minimum cost for each of them individually. The least delay tree (LD) is a tree originating at the source *S* and spanning all members of *M* with minimum delay for each of them individually.



(a) Original graph

(b) Least Cost Tree with Average Cost = 3, Average Delay = 6, Maximum Delay = 7

(c) Least Delay Tree with Average Cost = 7.5, Average Delay = 3.5, Maximum. Delay = 4

(d) Optimal DAG with Average Cost = 4.5, Average Delay = 5.5, Maximum Delay = 6.

Fig. 1 The difference between Least Cost Tree, Least Delay Tree and Optimal Directed Acyclic Graph for Minimum Shortest Path with Delay Constraint

The Optimal Delay Constrained Shortest Path Multicast Routing (ODCSP) problem is to find a route from source *S* and spanning all members of *M* satisfying the following conditions:

1 - Minimum *Cost (Path (S , v))* $\forall v \in M$

2 - *Delay (Path (S, v))* $< \Delta$ $\forall v \in M$

Where Δ is the maximum delay permitted for any destination. The example shown in Fig. 1 shows the difference between the three trees. Fig. 1.a contains the topology of the network, Fig 1.b and Fig 1.c contain the least cost/delay tree which can be easily achieved by applying Dijkstra algorithm on the cost/delay of links, it always returns the optimal tree in $O(n^2)$. Fig 1.d, shows the Optimal Delay Constrained Shortest Path Multicast Routing (ODCSP) problem, which is known to be NP-Complete. The figure shows that for ODCSP problem, the result can be represented as *Directed Acyclic Graph (DAG)*. So, to send packets from A to D, the packets follow the route ABD, while to send packets from A to E, the packets follow the route ACBE, this can be adjusted from the routing table at node A. In this paper we propose a new algorithm that finds a near optimal DAG for ODCSP problem.

3. Delay Constrained Shortest Path Multicast Algorithm (DCSP)

DCSP, its pseudo code is presented in the appendix, is a heuristic algorithm to find a near optimal DAG for delay constrained shortest path problem. DCSP is based on flooding, where it starts with a single token at the source node *S*. The token is continuously duplicated from node to node yielding a set of tokens where each token keeps track with the cost and delay it experiences so far, also it keeps track with the path from source node *S* to the node it currently belongs to. Tokens are collected at the destinations to determine the winner token which reaches with the least cost and satisfies the delay constraint Δ. The winner tokens from all destinations are combined together each with its path from source to destination to construct the required DAG.

To prevent the excessive increase of tokens in the network and hence the exponential execution time, we limit the number of tokens that can be concurrently in any node by the number *K* which can be ranged from 1 to *N* where *N* is |*V*|, the number of nodes in the network. This will control the duplication of tokens and assure a polynomial time execution. To achieve this, we consider four constraints tested for any token to be duplicated from node *u* to node *v* and if any one of them is satisfied we will not duplicate the token, the four constraints are:

1. The sum of the token's delay and the link delay between nodes *u, v* will exceed the delay constraint Δ. So, there is no need to complete the token's trip.
2. The token *T* visited node *v* before, so, it is going to make a loop. This can be achieved by checking whether node *v* stamped token *T* before or not.

3. There was a token $T`$ visited node $v$ before and it is better than token $T$. So, there is no use to continue token $T$ since any result come from it will be dominated by token $T`$.
4. Node v has already K tokens and there is no room for a new token, in this case we will choose a victim token from K+1 tokens. According to our heuristic, we will choose the one with maximum delay, this is because such a token has the least probability to continue its trip. Another point for choosing this token is that we need to guarantee finding a feasible path, so we have to keep the token with minimum delay.

Constraint 1 can be tested by keeping track of the token's delay by adding the delay of each link it passes. Constraint 2 requires that every node u that token passed put its stamp on the token. Constrains 3 and 4 require that we keep a history list with length K associated with each node to keep track with the best K tokens passed on this node so far. This list is sorted in ascending order w.r.t token's delay and in descending order w.r.t token's cost. Any two consecutive tokens $T`$ and $T``$ must have $T_D` < T_D``$ and $T_C` > T_C``$. Any new token $T$ should be compared to the history list and find its appropriate place w.r.t its delay, the place where $T_D` < T_D < T_D``$, then we look for T_C which can be one of the following three cases :

1. $T_C` < T_C$ , in this case token $T`$ is better than token T where it has less cost and less delay. So, we will not duplicate token T.
2. $T_C`` > T_C$, in this case token $T$ is better than token $T``$ where it has less cost and less delay. So, we will duplicate token T and delete token $T``$ from the list.
3. $T_C > T_C > T_C``$, in this case we can not favor any token to another, so we will duplicate token T and add it to the history list.

Lemma I: DCSP algorithm always finds a DAG if one exists and if DCSP fails to find it, then there is no other algorithm can find it.

Proof : The optimal solution for delay constrained shortest path multicasting problem can be found by exhaustively examining all the paths from source node S to each destination individually and select the best path for each one. In our algorithm, if we do not put any constraints in duplicating the tokens, then the tokens will increase exponentially and span all the possible paths in the network as in flooding, and we will find a token for each path which will permit us to choose the best one for each destination easily. The effect of the four constraints described above in the algorithm will be as follows: Constraints 1, 2 and 3 tends to cancel the tokens (paths) that will not be optimal in its early stages and hence reducing the complexity but without any effect on the optimality. Constraint 4, which is the heuristic constraint, kills the token with the maximum delay of K+1 tokens, there is no way to prove that the cancelled token will not be in an optimal path. But, this heuristic guarantees that we will find a solution if one exists, since even with K=1, the least value of K, we will have two tokens to choose one of them and, we will cancel the one with maximum delay and keep the other with least delay, then at this extreme case we will always keep track with the least delay token (path) that yields the least delay tree (LD) which means that there is no tree can give less delay for any destination than this tree. So, if LD tree does not satisfy the delay constraint, then there is no other tree can satisfy it. Since DCSP is reduced to LD algorithm when K=1, then if DCSP fails to find a feasible tree, then no other algorithm can find it.

Lemma 2: The worst case complexity of DCSP is $O(K^2N^2)$, where K is an integer value ranged from 1 to N and N is $|V|$ the number of nodes in the Network.

Proof : The algorithm is continuously looping till all the tokens in the network finish their trips, and since the maximum length of any trip is bounded by the number of nodes in the network N, then the algorithm will loop at most $O(N)$ times in duplicating tokens. For each time of duplication the algorithm checks all the N nodes for existence of tokens, so checking nodes will be $O(N^2)$. For each node, we will process each token in it and since there will be at most K tokens in each node then tokens will be processed $O(KN^2)$. For each token we will test whether it can be duplicated or not for all neighbor nodes, so, if we assume that the network has and average degree d, then token duplication will be tested $O(dKN^2)$. If, at the worst case, all the tested tokens will be duplicated and we insert the duplicated token in a sorted list with size K which can be achieved in $O(K)$ by insertion sort, then the whole algorithm can be executed in $O(dK^2N^2)$. But since we use real networks which always has a small node degree, we can consider that d is a small constant number, so the complexity of DCSP will be $O(K^2N^2)$.

4. Performance Analysis

4.1 Random Graph Generator

To guarantee fair simulation results, we use the same graph generator [14] that is used in all problems related to multicasting. N nodes are randomly distributed over a rectangular area with size 2000 × 2000 where each node placed at a location with integer coordinates. The probability of edge existence between any two nodes u, v can be calculated from the function:

$$P(u, v) = \beta \exp\left(\frac{-d(u, v)}{L\,\alpha}\right)$$

Where $d(u, v)$ is the distance between nodes u, v. L is the maximum distance between any two nodes. α and β are two parameters used to adjust the degree of the graph and the density of short and long edges. After calculating the above function for each pair, the resulted graph does not

necessary to be connected, so, we add random edges to obtain a connected graph. The cost of any edge *e(u, v)* equals to the distance *d(u, v)* and the delay is a random value between 1 and 10. Finally, for each algorithm to be correctly evaluated, we run it on 3000 different graphs and taking the average.

4.2 Simulation Results

We compare our algorithm DCSP with three different values of *K* against the least delay tree LD, that comes from applying Dijkstra algorithm, and CDKS algorithm [11] which comes from combining least cost tree and least delay tree. The optimal result is plotted in each graph to show how far each algorithm is from the optimal.

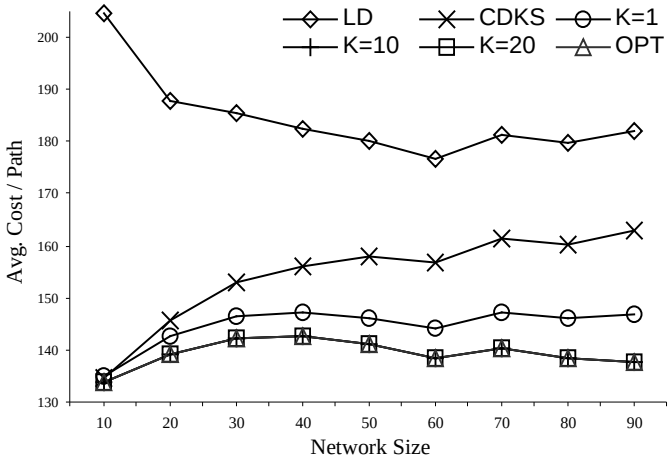


Fig. 2 Effect of network size with multicast group=10%, delay limit = 20, average degree = 8

In Fig. 2, we change the network size while keeping the delay limit=20, average degree=8 and the multicast group is 10% from the network size. It can be observed that DCSP is always dominating CDKS even when *K*=1. The three line of *K*=10, *K*=20 and OPT are almost identical which means that *K*=10 is sufficient to get the optimal solution in this case of delay limit, average degree and group size even when the network size increases. Also, as the network size increases the difference between DCSP and CDKS increases which means that DCSP performs better with the increasing of network size.

In Fig. 3, we set network size to 50 nodes, delay limit=20 and average degree=10, while changing the group size from 5 (10% of network) to 50 (broadcasting). The results show that, as previous, DCSP is always dominating CDKS even with *K*=1. *K*=5 is sufficient to get the optimal results. The performance difference between algorithms is almost constant with the increasing of group size, this can be deduced from the fact that CDKS and DCSP are based on Dijkstra and flooding algorithms which designed for broadcasting and then pruning the undesired links.

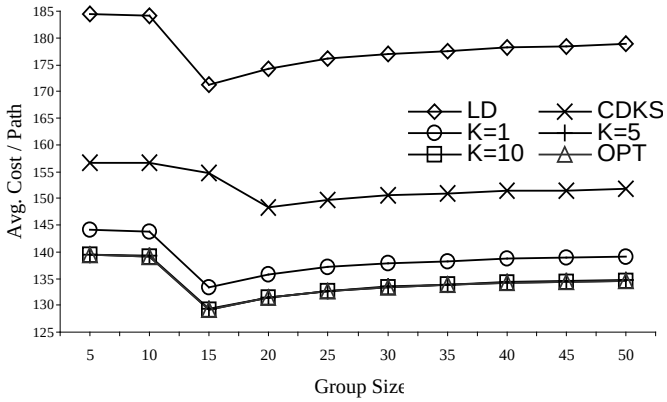


Fig. 3 Effect of multicast group size with network size = 50, delay limit = 20, average degree = 10

In Fig. 4, we also set network size to 50 nodes, delay limit=20 and group size=5. The average degree of the graphs is changed from 4 to 16 yields that for the small degrees CDKS gives the same results as DCSP with *K*=1 while with the increase of average degree DCSP dominated CDKS. Results also show that DCSP with *K*=10 gives the same results as OPT.

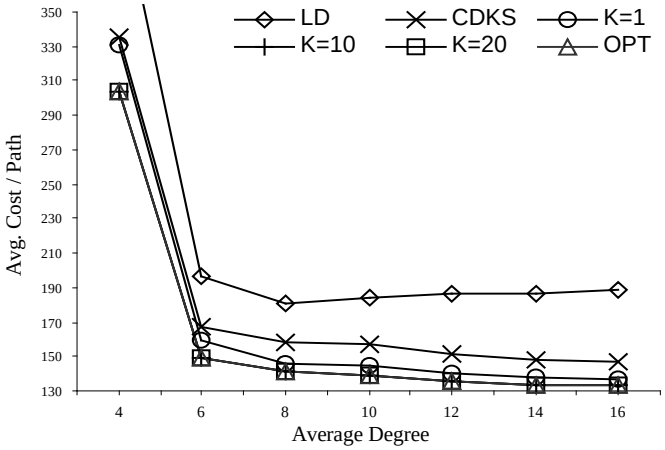


Fig. 4 Effect of average degree with network size = 50, multicast Group = 5, delay limit = 20

In Fig. 5 and Fig. 6, we investigate the effect of relaxing the delay constraint. In the two figures, we can see that for small delay limit, DCSP is always better than CDKS. As the delay limit relaxed, CDKS is going towards OPT this is because CDKS first computes the least cost tree and as the delay limit increased the problem tends to be finding the least cost tree without constraints. In Fig. 5, we use graphs with averaged degree=4 while in Fig. 6 we use graphs with average degree=10. This makes that in low

average degree it is sufficient for K=5 to get the optimal results and the average cost per path is much higher than in graphs with high average degree which need K=10 to get optimal results.

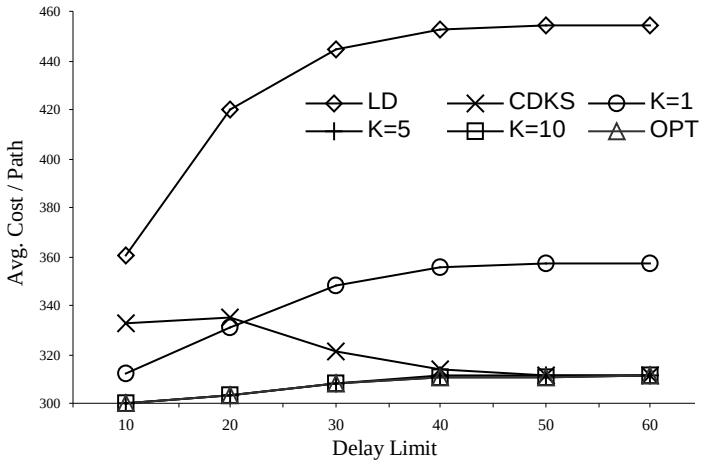


Fig. 5 Effect of delay limit with network size = 50, multicast group = 5, average degree = 4

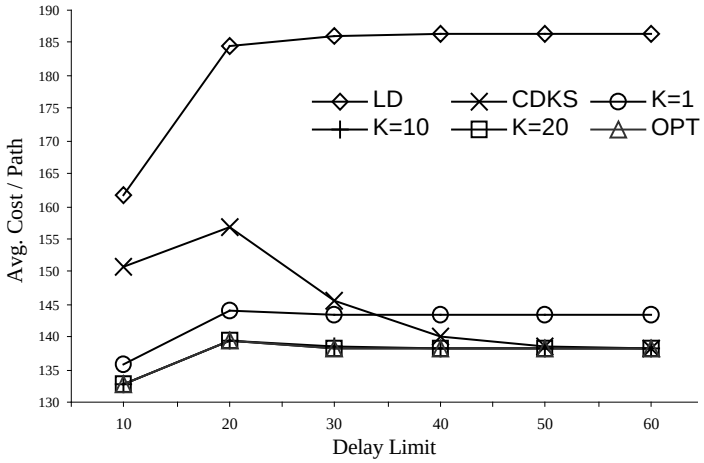


Fig. 6 Effect of delay limit with network size = 50, multicast group = 5, average degree = 10

In Fig. 7 and Fig. 8, an optimal value of parameter K is determined. Only DCSP with different values of K and OPT algorithms are simulated. The value of K is growing exponentially toward the optimal solution. In Fig. 7, with average degree=4, the optimal result is achieved at K=7, while in Fig. 8 with Average degree=8, the optimal result is achieved at K = 16.

5. Conclusion

In this paper we propose a polynomial time heuristic algorithm that computes the shortest path tree with delay constraint. The algorithm has a running time $O(K^2N^2)$ where K is a variable adjusted from 1 to N and N is the number of nodes in the network. Simulation experiments have been done to compare the efficiency of the new algorithm with other previous algorithms and with the optimal results. Empirical results show that our algorithm is always dominating previous algorithms and gives optimal results with certain value of K. Simulations are also done to determine the appropriate value of K that gives the optimal result. It is clear that a small value of K could be enough which makes the running time of the algorithm near $O(N^2)$.

The work in this algorithm can be extended in three ways. Firstly, a distributed version of this algorithm could be introduced by limiting the data kept in each node. Secondly, the dynamic change of group members should be considered to be embedded on the algorithm and not to start the algorithm from the beginning. Finally, this algorithm should be incorporated in an appropriate protocol to be used in real networks.

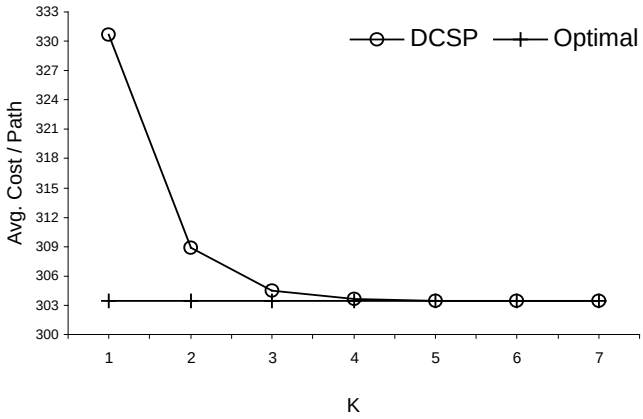


Fig. 7 Effect of K with network size = 50, multicast group = 5, delay limit = 20, average degree = 4

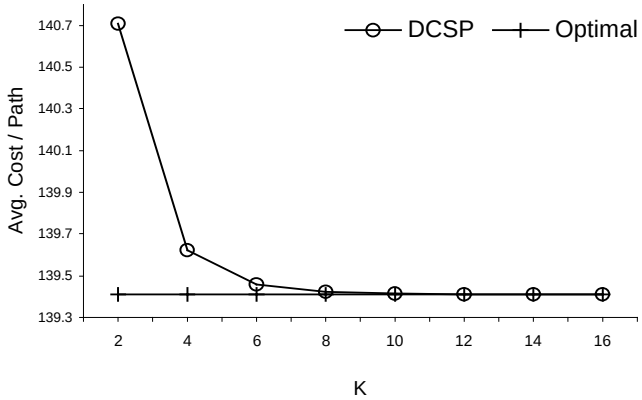


Fig. 8 Effect of K with network size = 50, multicast group = 5, delay limit = 20, average Degree = 10

References

[1] P. Winter, "Steiner Problem in Networks: A Survey", *Networks*, vol. 17, no. 2, pp. 129-167, summer 1987,

[2] F .Hwang and D. Richards, "Steiner Tree Problems", *Networks*, vol. 22, no. 1, pp 55-89, January 1992.
[3] R. Bellman, *Dynamic Programming*. Princeton University Press, 1957.
[4] J. A. Dossey, A. D. Otto, L.E. Spence,and C.V. Eynden *Discrete Mathematics*, Second Edition, Harper Collins College Publishers, 1993
[5] V. Kompella, J. Pasquale, and G. Polyzos, "Multicast Routing for Multimedia Communication", *IEEE/ACM Transactions on Networking*, vol.1, no. 3, pp 286-292,June 1993.
[6] V. Kompella, J. Pasquale, and G. Polyzos, "Multicasting for Multimedia Applications", *in Proceedings of IEEE INFOCOMM'92*, pp. 2078-2085, 1992.
[7] R. Widyono, "The Design and Evaluation of Routing Algorithms for Real-Time Channels", Tech. Rep. ICSI TR-94-024, University of California at Berkeley, International Computer Science Institute, June 1994.
[8] S. Wi and Y.Choi, "A Delay-Constrained Distributed Multicast Routing Algorithms," *in Proceeding of the twelfth International Conference on Computer Communication (ICCC '95)*, pp. 883-838, 1995.
[9] Q. Zhu, M. Parsa, and J. Garcia-Luna-Aceves, "A Source-Based Algorithm for Delay-Constrained minimum-Cost Multicasting", *in Proceeding of IEEE INFOCOM'95*, pp. 337-385, 1995.
[10] X. Jia, "A Distributed Algorithm of Delay-Bounded Multicast Routing for Multimedia Applications in Wide Area Networks" *IEEE/ACM Transactions on Networking*, Vol. 6, No.6, pp. 828-837, December 1998
[11] Q.Sun and H. Langendoerfre, "Efficient Multicast Routing for Delay-Sensitive Applications", *in Proceedings of the second Workshop on Protocols for Multimedia Systems (PROMS '95)*, pp. 452-458, October 1995.
[12] H. Salama, "Multicast Routing for Real-Time Communication on High-Speed Networks," PhD. Dissertion, North Carolina State University. Department of Electrical and Computer Engineering.
[13] H. Salama, D. S. Reeves and Y. Viniotis, "Evaluation of Multicast Routing Algorithms for Real-Time Communication on High-Speed Networks", *IEEE Journal on Selected Areas of Communication*, Vol. 15, No. 3, pp. 332-345 April 1997.
[14] B. Waxman, "Routing of Multipoint Connections", *IEEE Journal on Selected Areas of Communication*, Vol. 6, No. 9, pp. 1617-1622 December 1988.

Appendix

/* Delay Constrained Shortest Path Multicasting Algorithm, the inputs to the algorithm are:
1- Graph ***G*** with nodes ***V*** and links ***E***, ***G (V, E)***.
2- Source Node ***S*** 3- Multicast Group ***M***
4- Delay Constraint **Δ** 5– Constant ***K***
The algorithm outputs a ***DAG***, which contains all the links needed for routing in the shortest path. */

DAG **DCSP** (*G(V,E)* , *S* , *M* , Δ , *K*)
{
Insert a token in *S*
No. Of Tokens = 1
While (*No. of Tokens* > 0)
for each Node *v* in *V*
for each Token *T* currently in Node *v*
{ f**or each** neighbor Node *u* to Node *v*
{ **if Can_Duplicate** (*T, v , u*) Duplicate the token
if Node $u \in M$ update the winner Token so far.}
No. Of Tokens = No. Of Tokens – 1 }
for each Node *v* in *M*
Insert all links from *v*'s winner Token in *DAG*
Return *DAG*
} /* for **DCSP** */

/* Function **Can_Duplicate** takes the input :
1 - Token *T*
2 - The Node *v* that currently hold the token
3 – The Node *u* that the token wants to duplicate.
Then the function outputs either ***False*** when the Token *T* can't be duplicated from *v* to *u* or ***True*** when the token can safely be duplicated from *v* to *u* */

Boolean **Can_Duplicate** (*T , v , u*)
{
/* The Token will exceed the Delay Limit */
if $(T_D + Delay (v , u)) > \Delta$
Return *False*
/* The Token will make a loop */
if Node *u* stamped Token *T* before
Return *False*
/* We got a better token before */
/* Compare the current token by the history kept at node *u* to see if there was a better token in terms of delay and cost w.r.t current token */
if Node *u* got a better token before
Return *False*
/* There is no room for the new Token */
/* We limit the number of tokens that can be kept by any Node by the number *K*, So if we have a token after the first *K*, we have to choose a victim from the *K* +1 tokens, our heuristic is to choose the token with the highest delay */
if there are *K* tokens in Node *u*
if Token *T* has the highest delay among all tokens in node *u*
Return *False*
/* Now the Token *T* is valid for duplication, and we have to put it in the history of Node *u* as one of the best tokens so far */
If the history list of Node *u* has *K* entries
Delete the one with the highest delay
Insert Token *T* in the history list of Node *u*
Return *True*
} /* for **Can_Duplicate** */