Steiner Tree Compilation of Multicast under Differentiated Services Constraints

Stavros Vrontis, Stavros Xynogalas, and Efstathios Sykas

Abstract: **This paper deals with the creation of multicast trees in a differentiated services (DiffServ) domain. Initially, we model the integration problems of multicast & DiffServ and give a general description of our framework for multicast provisioning in DiffServ domains. Within this framework, we introduce a novel heuristic algorithm which calculates the multicast trees efficiently. The multicast tree's format and the bandwidth constraints per service class are modeled. The heuristic is based on the Dijkstra's shortest path algorithm and aims to produce the cheapest possible trees (Steiner tree problem) that conform to the defined model. The produced trees can be considered as DiffServ-customized Steiner trees. Furthermore, we evaluate the algorithm with theoretical and experimental analysis and finally, we present our conclusions.**

Index Terms: **Differentiated services (DiffServ), multicast, Steiner tree problem.**

I. INTRODUCTION

Multicast [1] is an efficient way of data packet delivery to multiple receivers. Bandwidth economy is achieved, since the unicast transmissions to the receivers are replaced with a single (multicast) transmission, where packet replication occurs only in specific nodes. Multicast is mostly applied to streaming applications, where the bandwidth demands are high (video conference, video on demand, etc.).

A directed multicast tree is formed during each multicast packet delivery session, starting from the source (root) and ending at the receivers (leaves). From the graph theory point of view, the problem of creating the tree with the lowest cost that spans a set of nodes in a weighted network graph is known as the Steiner tree problem [2]. The Steiner tree problem is NP-complete, hence effective solutions need to be based on heuristics.

The research community has been working towards the integration of multicast in state-of-the-art networks (e.g., mobile networks, ad-hoc networks, differentiated services (DiffServ) networks, etc.). Within this paper, we focus on the integration of multicast in DiffServ [3] networks, a particularly interesting case, because of the fundamental differences in the philosophy behind the multicast mechanism and DiffServ networks.

Specifically, in Section II, we discuss the problems that come up when trying to integrate multicast in a DiffServ domain. We briefly present proposed solutions by other researchers, as well as our own framework. In Section III, we model the problem of the calculation of QoS-aware multicast trees and present our heuristic algorithm, which aims to solve the *DiffServ-customized Steiner tree* problem. In this section, we also present the most important algorithms, which deal with the aforementioned problem. In Section IV, we compare our algorithm with the others and evaluate it by performing theoretical and experimental analysis. Finally, in Section V, we draw our conclusions.

Manuscript received February 11, 2005; approved for publication by Tony Lee, Division III Editor, May 19, 2006.

S. Vrontis is with the Vodafone Greece, Greece, email: stvront@telecom.ntua.gr.

S. Xynogalas is with the National Technical University of Athens, Greece, email: staxy@telecom.ntua.gr.

E. Sykas is with the Division of Communications, Electronics and Information Engineering, Department of Electrical Engineering, National Technical University of Athens, Greece, email: sykas@cs.ntua.gr.

II. MULTICAST IN DIFFSERV

Multicast packet delivery is difficult to be achieved in DiffServ domains. Within multicast, all the domain's nodes must maintain state information for the multicast groups (e.g., which network interface guides to one or more members of a multicast group). Hence, multicast is not scalable. On the other hand, DiffServ's main advantage, compared to other QoS provisioning architectures, is scalability. Scalability is achieved by pushing all complex functions (marking, policing, classification, etc.) to the edges of the network domain, while the core routers simply classify the aggregated traffic according to the value (DiffServ code point: DSCP) of the type of service (TOS) field of the IP header. Moreover, the "core simplicity" principle of DiffServ is violated within multicast, since all routers perform complicated operations that are required for multicast packet transmissions.

The main problem of multicast & DiffServ integration arises because of the unpredictable form of multicast trees. Once a new multicast member joins a multicast group, a new branch towards this node is added to the current multicast tree. The resources for this branch have not been reserved *a priori*, since it cannot be foreseen if and where the branch will be created. Hence, packets of the flow that traverses the new branch will be lost in case there is heavy network congestion (neglected reservation sub-tree: NRS problem [4]).

Note that Bless *et al.* present explicitly the NRS problem in physical language [4]. The reader is encouraged to read this document to get a detailed analysis with examples, while within our article we formulate the problem to complete the whole picture.

Another multicast & DiffServ integration problem arises due to the fact that the QoS provisioning in DiffServ is source-driven meaning that the packets are marked in the source's side (the source's neighbor edge router), while multicast is a receiver-driven procedure meaning that the procedure depends on the receivers' events (join/leave group). Hence, the service level for the traffic flow depends only on the source and no mechanism has been foreseen for service differentiation among the receivers (the heterogeneous QoS problem).

A. The Underlying Framework

Numerous solutions have been proposed for the integration of multicast with DiffServ, however none of them solves completely the problem, while sometimes they constitute new problems. The current solutions can be classified into three main categories; a) state-based, b) edge-based, and c) encapsulation-based.

The state-based solutions require all the domain's nodes to store state information per multicast group. The statelessness of the core routers is violated and poor scalability is achieved. The most important solutions in this category are the Bless' IETF-RFC [4], the QUASIMODO [5], DAM [6], QMD [7], and RE-UNITE [8] frameworks.

Within the edge-based solutions, multicast functions are limited to the edge routers. No trees are built and no packet replications are performed in the core network. Obviously, the achieved bandwidth economy is limited. The most typical algorithm of this category is the edge-based multicasting (EBM) [9].

The encapsulation-based solutions are based on the extension of the IP header of the multicast packets in order to include the tree topology. The routers parse the IP header and decide to replicate or not the incoming multicast packet, determine the number of copies and select the proper DSCP value of the DS byte for each copy. The total length of the IP header and therefore the total length of the multicast packet increases proportionally to the number of the receivers, causing scalability. The main representatives of this category are the DSMCast [10] and the XCAST [11] protocols.

Unfortunately, all solutions in these categories suffer from different disadvantages and only solve the problems partially. In [12], we presented a framework for the multicast provisioning in DiffServ domains analyzing issues like group & bandwidth management, inter-domain multicasting etc. Within the current article we do not repeat this analysis, but we highlight the main components of the framework instead.

Packet delivery within our framework is done with a mechanism similar to the QoS-aware multicasting in DiffServ domains (QMD) and REUNITE. Specifically, the multicast transmission is achieved through equivalent unicast transmissions between the source, the receivers and specific *key* nodes. In conventional multicast terms, these key nodes are the nodes of the multicast tree with degree more than one. The source sends the data in unicast packets to the first key node. Each key node replicates the received packets, marks them and sends them to its children keys via simple unicast transmission. Finally, the group members receive the data packets from the corresponding key nodes. The module at each key node performing these operations (namely, *multicaster*) assigns to each outgoing flow (directed to a subset of receivers) the DSCP value that corresponds to the highest service level for the receivers' requirements.

Fig. 1 demonstrates a multicast delivery via unicast transmissions: Receiver 2 (R2) requires S_i service level, receiver 4 (R4) requires lower level service S_{i-1}, and finally, receivers 1 (R1) and 3 (R3) require the lowest service level S_{i-2}. The source's (R0) unicast packets are received from multicaster-0 at node B. Multicaster-0 creates two flows: One flow is marked with the S_i DSCP and sent to multicaster-1 at node C and another is marked with the S_{i-1} DSCP and sent to R4. Multicaster-1 creates two flows towards the R1 and the multicaster-2 at node G and so on.

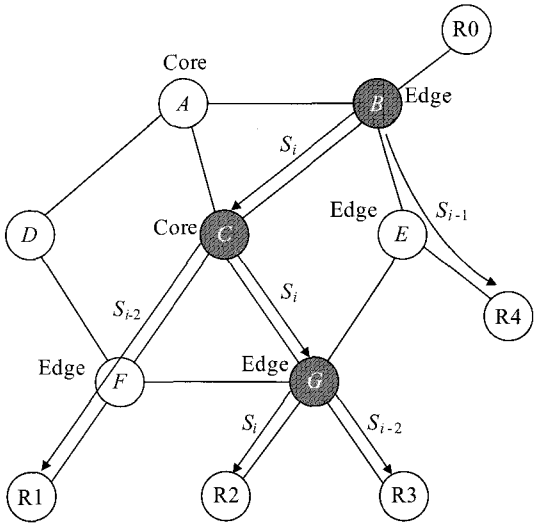


Fig. 1. Multicast equivalent with unicast transmissions.

Within traditional DiffServ, the bandwidth management is done by the domain's administrative component, namely the bandwidth broker (BB). The BB stores information for the consumed bandwidth at each link for each service class. Since the BB is aware of the available bandwidth resources of its domain, we extend its functionality with the calculation of the (equivalent) multicast tree. Hence, the NRS problem is solved, since the BB predicts the bandwidth consumptions at all links per service class before "producing" the multicast tree. Within the current article, we focus on the calculation of the multicast tree. The multicast tree should comprise of the least possible links, and obviously have the format that was discussed earlier (constrained Steiner tree problem).

III. CALCULATING QOS-AWARE MULTICAST TREES

Within this section, we focus on the algorithm we propose for the calculation of QoS-aware multicast trees. The algorithm is specific and is applied for our framework, which was briefly introduced previously.

A. Problem Modeling

Using the common notation for graphs [13], we denote the links' cost function c: $E \rightarrow \Re$ as c_{ij} for the link e_{ij}. For a network H, the notation $c(H)$ corresponds to the sum of the costs of all links in H. The set of the terminal nodes is denoted as

$$R = \{z_1, z_2, \cdots, z_r\}, \text{ where } r := |R|.$$

The z_1 terminal is the source, while the set of the receivers of z_1 is denoted as

$$R^{z_1} := R \backslash \{z_1\}.$$

Moreover, the set of the receivers of z_1 with required service level s is denoted as $R_s^{z_1}$ ($R^{z_1} = \bigcup_s R_s^{z_1}$).

A cut in a graph $G(V, E)$ is defined as a partition $C = (\overline{W}, W)$ of V ($\emptyset \subset W \subset V$; $V = W \cup \overline{W}$). We use $\delta(W)$ to denote the set of links of E that $u_i \in \overline{W}$ and $u_j \in W$ or $u_i \in W$

and $u_j \in \overline{W}$ (for undirected graphs). For simplicity, we write $\delta(u_i)$ instead of $\delta(\{u_i\})$. A cut $C = (\overline{W}, W)$ is a Steiner cut if $R \cap W \neq \emptyset$ and $R \cap \overline{W} \neq \emptyset$.

The classical Steiner tree problem is defined as following: Given a network $G(V, E)$ and a non-empty set of terminals R ($R \subseteq V$), find a tree $T_G(R) \subseteq G$, such that there is a path between any pair of terminals and $c(T_G(R))$ is minimized. The Steiner tree is an acyclic, connected sub-network of G, spanning (a superset of) R.

Within this paragraph, we give the formulations in order to define the problem of finding the multicast tree in DiffServ networks with the minimum cost. This problem is a constrained Steiner tree problem with bandwidth constraints for each link per service class. Moreover, we have that $c_{ij} = 1$. The goal is to find the tree $T \subseteq G$ that is described in Section II with the minimum cost, without violating the mentioned constraints.

We use the following binary variable X_{ij_s} for each edge, indicating whether an edge is part of the solution ($X_{ij_s} = 1$) or not ($X_{ij_s} = 0$). The pointer s specifies the service level for the traffic that traverses the (u_i, u_j) link. Specifically, the X_{ij_s} variable is defined as following:

$$X_{ij_s} = \begin{cases} 1, & \text{if } e_{ij} \in T, \text{ and service level of } u_i \rightarrow u_j \text{ is } s \\ 0, & \text{otherwise.} \end{cases} \tag{1}$$

Obviously, we have

$$\sum_s X_{ij_s} \leq 1. \tag{2}$$

We will use the following notations:

$$X_k(S) = \sum_{e_{ij} \in S} X_{ij_k} \text{ and } X(S) = \sum_k X_k(S).$$

We state the problem using the following formulations:

$$\sum_{s=1}^{s_{\max}} \sum_{\forall e_{ij} \in E} X_{ij_s} \rightarrow \min \tag{3}$$

$$(b^c_{ij_s} + r_{S_T}) X_{ij_s} \leq b_{ij_s}, \ \forall s > 1, \ \forall e_{ij} \in E \tag{4}$$

$$X(\delta(W)) \geq 1 \tag{5}$$

$$s \geq l, \ \text{ if } X_{ij_s} = X_{jk_l} = 1; \ |P_T(u_i, z_1)| < |P_T(u_j, z_1)| \tag{6}$$

$$\sum_{k=s}^{s_{\max}} X_k(P_T(z_1, z_i)) = |P_T(z_1, z_i)|, \quad \forall z_i \in R_s^{z_1} \backslash \{z_1\} \tag{7}$$

$$X_{ij_s} \in \{0, 1\}^{|E|}. \tag{8}$$

The calculation of the X_{ij_s} variables for the above linear system of formulations provides the problem's solution. Equation (3) represents the objective function, which is the total number of links of the multicast tree, while (4) represents the bandwidth constraints at each link for each service class. Notice that the constraint is not valid for $s = 1$ (e.g., for the best effort class), as appears in the formulation. The restriction (5) guarantees that in any set of links corresponding to a feasible solution, there is a path from z_1 to any other terminal. Equations (6) and (7) guarantee that the multicast tree's structure conforms to the one defined in the previous section. Specifically, the formulation (6) states the service degradation across the tree: Walking across the tree, the service level of the traffic that traverses the tree is degraded while we are moving towards receivers with lower service level requirements. Equation (7) guarantees the transmission of traffic with the required service level to each receiver. (7) may not be possible together with the formulation (4) (e.g., there is no path with the available bandwidth for the required service class for a receiver) and in such case there will be no feasible solution to the problem. However, note that within the proposed heuristic algorithm (see next section), we move each user (that requires service level s) from the set $R_s^{z_1} \backslash \{z_1\}$ to the lower service level set $R_{s-1}^{z_1} \backslash \{z_1\}$ if there is no path with the available bandwidth for the required service class (for $s > 1$). This way the multicast packet delivery is guaranteed for all receivers in case of congestions, degrading, however, the service level. We remind here that the constraint (4) is valid for $s > 1$. Therefore, our proposed algorithm always provides a feasible solution, since (4) is careless for the receivers of $R_1^{z_1} \backslash \{z_1\}$.

The problem is a NP-complete problem. By taking $s = s_{\max} = 1$, (3)–(8) define the classical Steiner tree problem with bandwidth constraints (NP-complete). Additionally, if we remove (4) the remaining formulations correspond to the classical Steiner tree problem (NP-complete).

B. Multicast Tree Calculation Algorithms

Several algorithms have been proposed for the calculation of multicast trees. The most significant of which are the shortest path tree (SPT), M-QOSPF [14], QMRP [15], [16], QMD-DIJKSTRA [17], spanning-joins protocols [17], and QoSMIC [18], [19].

The SPT algorithm (used in core based tree (CBT) [20] and protocol independent multicast (PIM) [21]) joins the new member with the root of the multicast tree, finding the shortest path towards the root. In this way, the produced tree is not necessarily the least-cost tree.

The M-QOSPF is based on the QoS routing extensions to OSPF and it assumes that all the routers know the domain's topology as well as the QoS status. When a new router wants to join a multicast group, it computes a QoS-satisfied path toward the source (or core router) and sends join request to the source along the path.

QMRP is a distributed QoS-aware multicast routing algorithm. In QMRP, a new member sends a signaling message along its shortest-path route towards the source in order to check each link to see if it has the available resources. If not, QMRP backtracks to the previous hop. The message is copied and detoured to the adjacent nodes, which try to continue following the shortest-path route to the source. The procedure is repeated until one or more messages reach the source. Where two feasible paths meet, a node must decide which path to use. Even though, exploring an unlimited number can lead to a high success rate, the overhead is high. Thus, a restricted form of QMRP has been proposed (QMRP-n), meaning that n nodes between the new member and the source can be actively detouring a signaling message.

In QMD-DIJKSTRA, the new member is connected to the

tree taking into account the QoS level per branch. The new member is attached to the tree with the shortest path to the closest node of the tree, which is being traversed by traffic with the required QoS level. The algorithm is QoS-aware and the current tree structure is exploited.

In the spanning-joins protocol, a new member broadcasts join-request messages in its neighborhood to find on-tree nodes. The on-tree nodes reply to the new member. The path of the reply message, determined by the unicast routing algorithm, is a candidate path. The new member may collect multiple reply messages, which contain QoS related information regarding the path. The new member selects the best candidate path. Consecutive broadcasts are necessary to search increasingly larger neighborhoods until on-tree nodes are found. This process can increase the overhead significantly.

In QoSMIC protocol, the search for candidate paths consists of two procedures, local search and tree search. The method is similar to the previous one, with the difference that the tree search allows QoSMIC to restrict its flooding local search in a small neighborhood.

C. MTCA

The proposed heuristic multicast tree calculation algorithm (MTCA) is based on the Dijkstra's shortest path algorithm [22] for the computation of the shortest path between the source and the destination. MTCA aims to produce the minimum-cost tree, which connects the source with the group's members.

MTCA takes as input the available bandwidth table $B = [B_{S_1} \quad B_{S_2} \quad \cdots \quad B_{S_{SMAX}}]$ and the requested transmission rate R, where B_{S_i} is the column table for the available traffic for the S_i service class.

Step 0: Initially, the corresponding cost table

$$C = [C_{S_1} \quad C_{S_2} \quad \cdots \quad C_{S_{SMAX}}]$$

is created from table B:

$$C_{ij} = \begin{cases} NORMAL\_COST, & \text{if } R \leq B_{ij} \\ INFINITE\_COST, & \text{if } R > B_{ij}. \end{cases}$$

The rule is *"assign the $NORMAL\_COST$ value (e.g., 100) to a link, if the requested rate is less than the available link's bandwidth for the specific service class, otherwise assign the INFINITE_COST value (e.g., 100000) to it."* Note that:

$$INFINITE\_COST \gg NORMAL\_COST.$$

For example, for transmission rate $R = 1$ Mbps, the BB will create the Table 2 for the case of Table 1 ($NORMAL\_COST = 100$, $INFINITE\_COST = 100000$).

Step 1: The receivers are classified into groups, which correspond to the service classes.

Step 2: The algorithm parses the receivers of each group successively. Specifically, it parses first the receivers of the highest-service-level group and in the end the receivers of the lowest-service-level group. Step 2 requires for each receiver the execution of the sub-steps 1 and 2.

Table 1. An example "links-available bandwidth" table.

Link	S_1 class	S_2 class	S_3 class
1	2 Mbps	1 Mbps	3 Mbps
2	0.1 Mbps	1 Mbps	5 Mbps
3	1 Mbps	0.2 Mbps	3 Mbps
4	2 Mbps	1 Mbps	2 Mbps

Table 2. The cost table for the case of Table 1 and $R = 1$ Mbps.

Link	S_1 class	S_2 class	S_3 class
1	100	100	100
2	100,000	100	100
3	100	100,000	100
4	100	100	100

Sub-step 1: Dijkstra's algorithm calculates the shortest path for a receiver utilizing the cost table C, which gives the links' costs. The selected path will not contain links that don't have the required available bandwidth. If the shortest path contains a link with cost equal to the *INFINITE_COST*, the receiver will be removed to a lower service level group (except from the receivers of the lowest service class). The cost table is updated after the calculation of the shortest path for the receiver as following:

$$C_{ij} = \begin{cases} MIN\_COST, & \text{if link } i \text{ belongs to path} \\ \text{unchanged}, & \text{otherwise.} \end{cases}$$

The MIN_COST value (e.g., 60) is assigned to the path's links for all classes. The updated cost table will cause the BB to prefer paths that contain links that have already been selected in previous steps. The selection of the MIN_COST value influences the produced multicast tree (see Section IV). Note that:

$$MIN\_COST < NORMAL\_COST.$$

Sub-step 2: The selected path is merged with the current tree.

The algorithm finalizes when all the receivers (of all groups) have been parsed.

IV. EVALUATION

Within this section, we evaluate the proposed algorithm by comparing it with related algorithms introduced in the previous section.

CBT is not QoS-aware and the produced trees are not least-cost by default, while our algorithm is QoS-aware and its target is to produce least-cost trees. Hence, CBT cannot be used in DiffServ domains.

The bandwidth consumed from the spanning-join and QoSMIC protocols is much more than the one consumed from our algorithm since these algorithms use flooding in order to find the on-tree nodes. Moreover, the algorithms aim to connect the new member with the on-tree nodes. This way, the produced tree is not necessarily the least-cost tree (a reconstruction of the

tree may produce a cheaper tree) and there might not be a QoS-satisfying path between the new member and the on-tree nodes, while there might be a QoS-satisfying path between the root and the new member. Hence, these algorithms do not always provide a solution; while MTCA always converges finding the QoS-satisfying path if there is one. If there is not such path, MTCA will still provide solution by degrading the new member's QoS requirements in order to connect him to the multicast tree.

M-QOSPF violates the basic principle of DiffServ, since it requires the core routers keep status information regarding domain's topology and QoS. Moreover, the algorithm calculates a QoS-satisfying path towards the source, without considering exploiting the current multicast flow. This way a total new branch may be created consuming unnecessary bandwidth resources, while a shorter branch could provide to the new member connectivity with the multicast tree.

Likewise M-QOSPF, QMRP requires the core routers keep status information regarding available resources violating the basic principle of DiffServ. QMRP will always find a QoS-satisfying path, if there is one; however, QMRP's main drawback is bandwidth consumption. A lot of bandwidth is consumed by the signaling messages, which actually test paths for available resources. Moreover, bandwidth economy is not considered since the structure of the current tree is not exploited and the algorithm does not aim to produce cheap trees.

Within QMD-DIJKSTRA, a new branch is added to the current multicast tree, when a new user joins the multicast tree. Even though the new branch is efficiently selected, adding branches to the tree does not lead to an optimal solution by default. The cost of a multicast tree calculated this way is generally greater than the one produced within MTCA, even though the signaling requirements are greater within our framework, since the multicast tree is cheaper in terms of bandwidth consumption and achieves the main goal of bandwidth economy. Moreover, the QMD-Dijkstra does not always provide solution since it aims to join the new member with the existing tree under QoS requirements, which may not be possible. This may be possible if the tree is restructured (MTCA).

Within the next paragraphs, we study the performance and the required time of MTCA. Performance refers to the number of links that constitute the multicast tree. Optimal performance is achieved if this value is equal to the minimum possible. This is essential because the less links constitute the multicast tree the less bandwidth is consumed. Moreover, the required time for MTCA is a crucial parameter for the overall system's performance. Specifically, it is more important for the case that users frequently join/leave the multicast group and consequently MTCA is executed many times. Hence, it is important that MTCA spends the least time possible. We will proceed with both theoretical and experimental analysis.

A. Theoretical Analysis

A.1 Performance Analysis

Within this paragraph, we calculate lower bounds for the performance of MTCA. The calculated limits define the maximum value of the ratio between the cost of the multicast tree (produced by MTCA) C and the theoretically optimal cost of the multicast tree C^*.

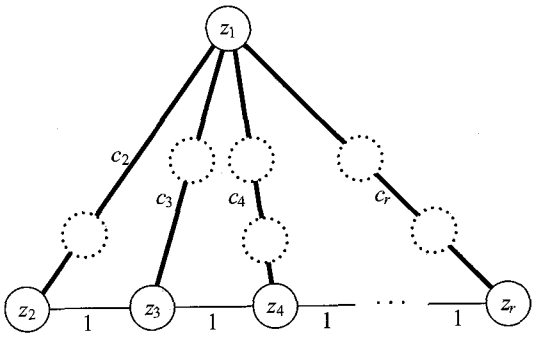


Fig. 2. Worst possible tree provided by MTCA.

We consider the worst case scenario concerning the MTCA's performance that appears in Fig. 2.

This is the case where all the calculated shortest paths (by MTCA) towards each receiver cannot be merged. Hence, a tree is created (bold lines) with $r-1$ branches, all of them starting from the source (z_1). The c_i value corresponds to the total cost of the branch $(z_1,\ z_i)$. The total cost C of the multicast tree is given by

$$C = c_2 + c_3 + \cdots + c_r. \tag{9}$$

In the above figure, we can see that the optimal tree's cost is: $C^* = r-1$, provided that z_1 is connected directly to at least one receiver (e.g., $c_j = 1$). Obviously, this value is the minimum possible for every network case with $r-1$ receivers. We have for MTCA that

$$MIN\_COST = k \cdot NORMAL\_COST \tag{10}$$

where k is a constant with $k < 1$.

Hence, when the first branch $(z_1,\ z_2)$ is selected, the cost for each link of this branch is multiplied with the k constant. Therefore, MTCA will apply the Dijkstra's shortest path (Dijkstra-SP) algorithm considering that the links of $(z_1,\ z_2)$ have now decreased cost value. Having selected for the receiver z_3 the path $(z_1,\ z_3)$ appearing in Fig. 2, we can see that being the shortest path, the selected path is also shorter than the path through z_2. Therefore, the following inequality is true: $c_3 < k \cdot c_2 + 1$.

Similarly, we can take for each receiver the following inequalities

$$c_4 < k \cdot c_3 + 1,\ c_5 < k \cdot c_4 + 1, \cdots,\ c_r < k \cdot c_{r-1} + 1.$$

We add the above inequalities

$$c_3 + c_4 + \cdots + c_r < k(c_2 + c_3 + \cdots + c_{r-1}) + r - 2.$$

For $k < 1$,

$$C < \frac{r - 2 + c_2 - k \cdot c_r}{1-k}. \tag{11}$$

The ratio between C and C^* is bounded by

$$C/C^* < \frac{r - 2 + c_2 - k \cdot c_r}{(r-1)(1-k)}.$$

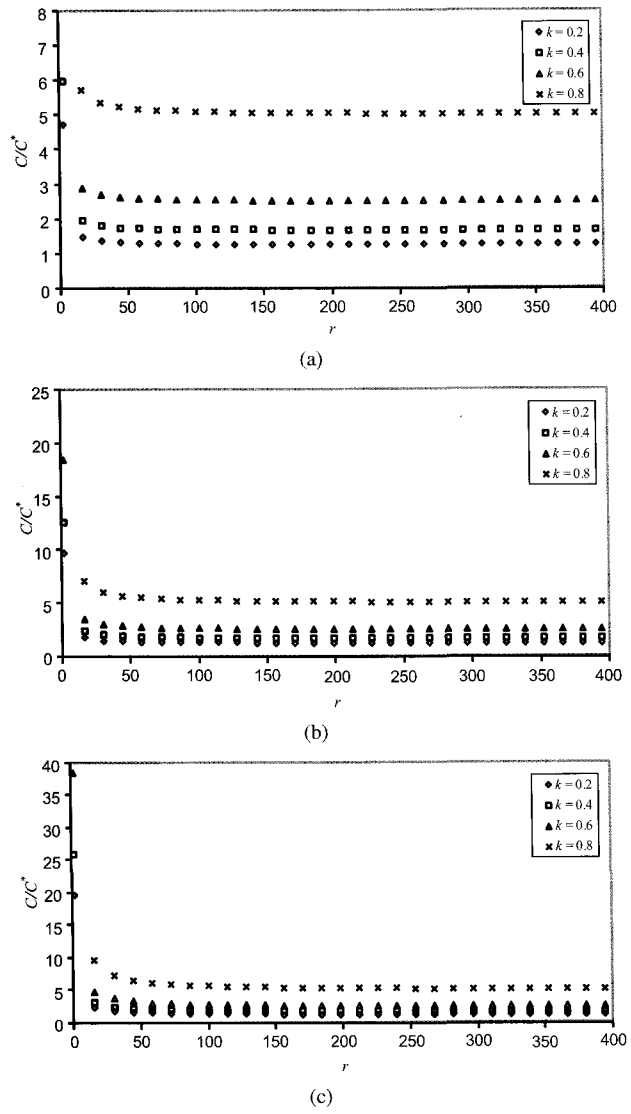


Fig. 3. Bounds for MTCA performance for three cases of network's diameter D: (a) $D = 4$, (b) $D = 8$, (c) $D = 16$.

If $c_2 = D, c_r = 1, C^* \geq 1$,

$$C/C^* < \frac{1}{1-k} + \frac{D-k-1}{(r-1)(1-k)} \tag{12}$$

where D is the network's diameter. For a large number of receivers, (12) leads to

$$\lim_{r \to \infty} C/C^* < \frac{1}{1-k}. \tag{13}$$

Equation (12) provides an upper bound (worst case scenario) for the ratio between C and C^*. We can see that the bound for the C/C^* ratio depends on the number of receivers and the network's diameter. Specifically, the bound decreases while the number of the receivers increases. On the other hand, the bound increases proportionally with the network's diameter.

In Fig. 3, we present (12) taking three cases for network's diameter. We can see that the network's diameter practically influences the bound only in case of small number of receivers. On the other hand, the k constant affects this bound significantly, as it also shown in (12) and (13).

Table 3. Five networks selected from the SP test-set of the SteinLib library.

Name	$\|V\|$	$\|E\|$	$\|r\|$	C^*
oddwheel3	7	9	5	5
antiwheel5	10	15	6	7
w13c29	783	2262	407	507
w23c23	1081	3174	553	692
w3c571	3997	10278	2285	2854

A.2 Time Analysis

In order to calculate the MTCA's required time, we examine the required time for each step analyzed in subsection III-C. Within step 2, the Dijkstra-SP algorithm is executed for each receiver to calculate the shortest paths. The running time of the Dijkstra-SP algorithm (the version that uses a Fibonacci heap for the storage of the vertices' labels) is $O(E + V \log V)$, where E is the number of edges and V is the number of vertices. Moreover, the cost table is sorted alphabetically. Since the (binary) search for a link requires $O(\log E)$, the update of the cost table requires, in the worst case, $O(D \log E)$ for each receiver (recall that D is the network's diameter). The merging of a path with the current tree requires $O(D)$ (worst case). Note that this period decreases, as the network graph becomes denser. Summarizing, it is easy to see that the required time for MTCA, in a graph with diameter D, V vertices, E edges, and r receivers is

$$O(r(E + V \log V + D \log E)). \tag{14}$$

The above formula shows that the MTCA's required time depends on the number of the receivers, the number of edges/nodes of the network graph and finally on the network's diameter.

B. Experimental Analysis

We performed tests to evaluate MTCA and crosscheck the theoretical studies of previous paragraph using network instances mostly selected from the SteinLib library [23]. SteinLib is a collection of Steiner tree network problems. A Steiner tree network problem is defined from the exact network topology (nodes and links) and the nodes of the network that are to be connected. Within SteinLib, the optimal solution or the best solution found so far is included for each Steiner tree instance. This information was used in order to evaluate the MTCA's performance.

Note that the experiments described in this section that are applied to instances from the SteinLib library refer to the five networks selected from the SP test-set (Table 3).

Besides SteinLib instances, we tested MTCA using custom network graphs created using the Waxman model [24]. The Waxman model randomly distributes nodes over a rectangular coordinate grid. The Euclidean metric is then used to determine the distance between each pair of nodes. Two nodes are connected with a probability $P(u, v)$ that depends on their distance:

$$P(u, v) = \beta e^{-\frac{d(u,v)}{L \cdot \alpha}} \tag{15}$$

where $d(u, v)$ is the distance from the node u to v, L is the maximum distance between two nodes, and α and β are parameters in the range $(0, 1)$.

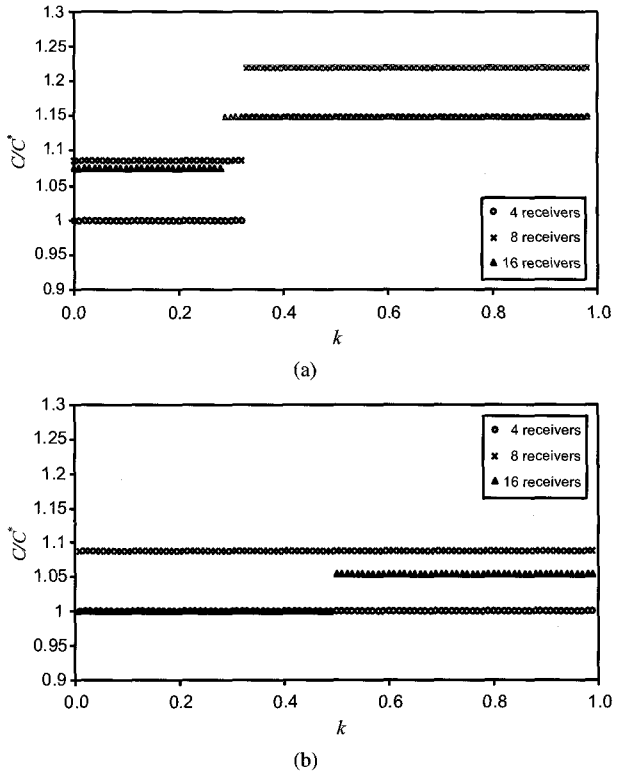


Fig. 4. (a) The MTCA has the maximum performance for $k < 0.33$, (b) for another network, we should select $k < 0.5$.

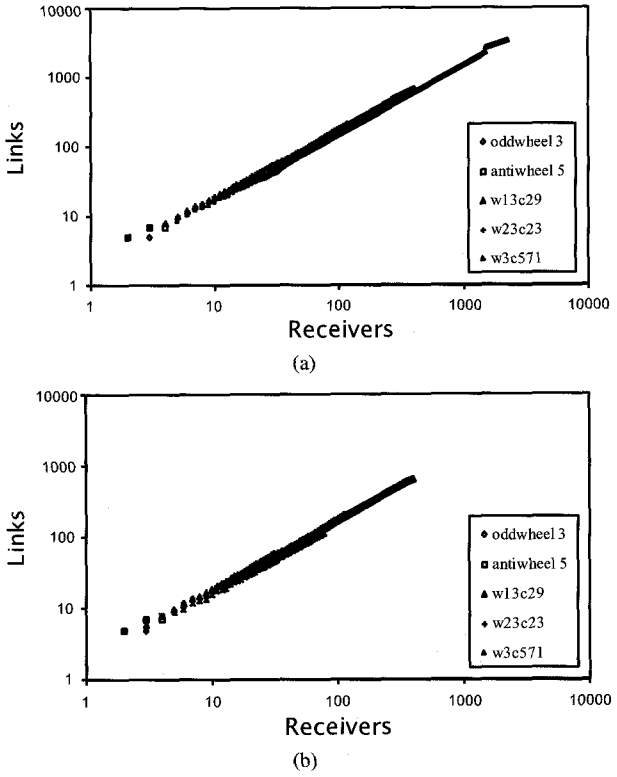


Fig. 5. The number of links of the produced multicast trees for SteinLib networks: (a) $k = 0.5$, (b) $k = 0.2$.

Finally, we mention that all tests were performed in a Microsoft Windows PC with CPU speed 1.4 GHz (Centrino technology) and 1024 MB RAM memory.

B.1 Performance Analysis

Theoretical analysis showed that MTCA's performance depends on the number of the receivers, the network's diameter and the value of the constant parameter k. We remind that k stands for the ratio of the $MINIMUM\_COST$ value with the $NORMAL\_COST$ value.

Initially, we will examine the influence of k to the MTCA's performance. We define as k_{opt} the values of k that give multicast trees with the minimum number of links. By performing tests for several network topologies and receivers' scenarios, we concluded that generally there is a threshold K for k values, which influences the path selection. Specifically, we have $\exists k \leq K$: $k = k_{opt}$. The tree's total cost is stable for $k > K$ and is improved for $k \leq K$. However, while the k_{opt} values provide trees with less total links, the length of the longest path from the source to any destination increases. Even though in some experiments the algorithm calculated the same multicast tree for all k values (e.g., $K = 1$), we observed the threshold's existence in most of the cases. The experiments showed that the threshold's value depends on the network topology. Generally, we concluded that $K \leq 0.5$. K depends on the order, in which the algorithm parses the receivers and on the receivers' position in the network.

Fig. 4 shows the results of two experiments in different network topologies. Both networks comprised of 40 nodes, randomly connected (Waxman grid). For each network we performed the experiment for 4, 8, and 16 receivers, modifying the value of k (e.g., the MIN_COST value).

Table 4. C/C^* ratio for the SteinLib networks.

Name	C/C^* $(k = 0.5)$	C/C^* $(k = 0.2)$
oddwheel3	1	1
antiwheel5	1	1
w13c29	1.33	1.28
w23c23	1.37	1.33
w3c571	1.2	1.18

Furthermore, we applied MTCA to the SteinLib instances for two cases; a) $k = 0.5$ and b) $k = 0.2$. MTCA was applied to numerous subsets of the receivers. Initially, MTCA experiments included only the first receiver from the SteinLib network. Then, we increased the number of the receivers, by including the next receiver and repeated the experiment. The procedure was repeated and ended when all of the receivers were included. The number of links for the produced multicast trees appears in Fig. 5 (logarithmic scale for both axes).

The values for the ratio C/C^* for the SteinLib networks (with the maximum number of receivers) appear in Table 4. We see that MTCA has excellent performance for small networks, while the performance decreases for larger networks.

The experiments with random Waxman grids gave max C/C^* ratio values close to 1.4.

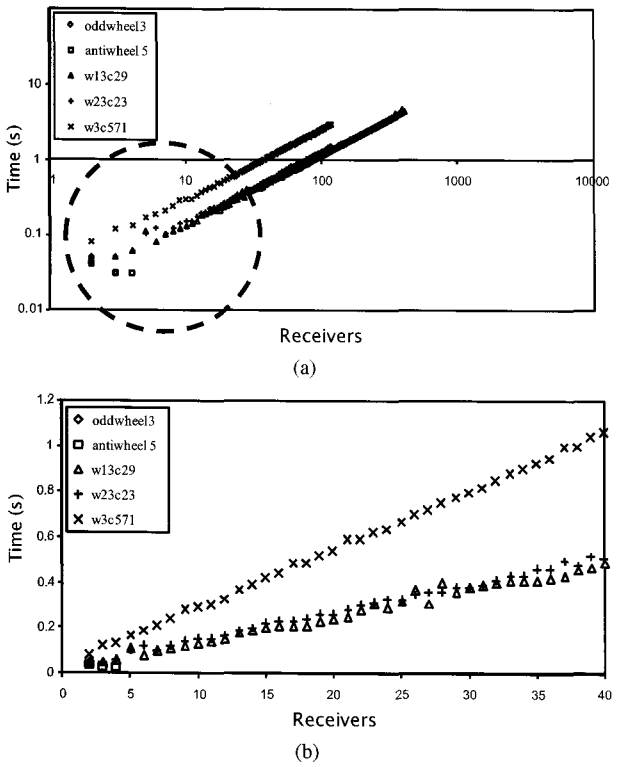


Fig. 6. MTCA's required time for SteinLib networks: (a) The calculation time for the SteinLib networks, (b) a magnification of the lower-left corner of the (a).

B.2 Time Analysis

We performed tests for several Waxman grids and SteinLib networks, in order to measure the required time for MTCA. The tests confirmed that the calculation time increases, while the number of receivers or the network's nodes/edges increases. The calculation time for the SteinLib networks appears in Fig. 6(a) (logarithmic scale for both axes). And Fig. 6(b) is a magnification (linear scale) of the lower-left corner of the first figure.

We conclude that the required time for MTCA is less than one second in normal cases (regarding the number of group members), a few seconds in case of larger networks, while one minute is required in extreme cases (huge networks).

V. CONCLUSIONS

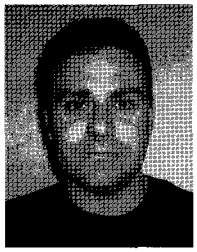
Within this paper, we dealt with the difficult problem of multicast in DiffServ domains. Initially, we formulated the problem instead of only describing it in physical language. Even though, this issue is explicitly analyzed in literature, a mathematical approach was missing. Then, we presented some features of our framework, which aims to solve the aforementioned problem. This solution requires the creation of multicast trees of a specific format. First of all, links with not enough available bandwidth for a specific service class should not be included in the produced multicast tree. Moreover, each receiver should be served with the requested service level, if this is possible. The idea of service degradation while moving towards the receivers should be reflected to the produced multicast tree. Of course, the number of links that comprise the multicast tree should be the least possible (Steiner tree problem). These restrictions were presented through mathematical formulations.

A novel heuristic, namely MTCA, was introduced to provide solutions conforming to these formulations. MTCA is explicitly described and evaluated via theoretical and experimental analysis. Within both methods, we evaluated MTCA's performance in terms of cost (number of links) and required time of the produced multicast trees. Experiments showed that MTCA provides trees with low cost and requires little time (less than 1 second) for small and medium network instances. Moreover, having that MTCA always converges (provides a solution in any case), we consider it as the optimal solution for the calculation of multicast trees suitable for our framework.

REFERENCES

[1] K. C. Almeroth, "The evolution of multicast," *IEEE Network*, vol. 14, no. 1, pp. 10–21, Jan./Feb. 2000.
[2] J. Ganley. The Steiner Tree Page. [Online]. Available: http://ganley.org/steiner
[3] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss. (1998, Dec.). Network Working Group: An achitecture for differentiated services. IETF. [Online]. Available: http://www.ietf.org/rfc/rfc2475.txt
[4] R. Bless and K. Wehrle. (2004, Apr.). IP multicast in differentiated services (DS) networks. IETF. [Online]. Available: http://www.ietf.org/rfc/rfc3754.txt
[5] G. Bianchi, N. Blefari-Melazzi, G. Bonafede, and E. Tintinelli, "QUASI-MODO: Quality of service-aware, multicasting over diffserv and overlay networks," *IEEE Network*, vol. 17, no. 1, pp. 38–45, 2003.
[6] B. B. Yang and P. Mohapatra, "Multicasting in differentiated service domains," in *Proc. IEEE GLOBECOM 2002*, 2002.
[7] Z. Li and P. Mohapatra, "QoS-aware multicasting in DiffServ domains," in *Proc. Global Internet Symp. 2002*, 2002.
[8] I. Stoica, T. S. E. Ng, and H. Zhang, "Reunite: A recursive unicast approach to multicast," in *Proc. IEEE INFOCOM 2000*, Mar. 2000.
[9] A. Striegel, A. Bouabdallah, H. Bettahar, and G. Manimaran, "EBM: Edge-based multicasting in DiffServ networks," in *Proc. Network Group Commun. (NGC)*, Sept. 2003.
[10] A. Striegel and G. Manimaran, "DSMCast: A scalable approach for DiffServ multicasting," *Computer Netw. J.*, vol. 44, no. 6, pp. 713–735, Apr. 2004.
[11] R. Boivie, N. Feldman, Y. Imai, W. Livens, D. Ooms, and O. Paridaens. (2003, Aug.). Explicit multicast (Xcast) basic specification. IETF Internet Draft. [Online]. Available: http://www.ietf.org/internet-drafts/draft-ooms-xcast-basic-spec-05.txt, work in progress.
[12] S. Vrontis and E. Sykas, "Extending differentiated services architecture for multicasting provisioning," *Computer Netw. J.*, vol. 48, no 4, pp. 567–584, Jul. 2005.
[13] S. V. Daneshmand, *Algorithmic Approaches to the Steiner Problem in Networks*, Ph.D. Dissertation, Mannheim, 2003.
[14] R. Guerin, A. Orda, and D. Williams, "QoS routing mechanisms and OSPF extensions," IETF Draft, Nov. 1996.
[15] S. Chen, K. Nahrstedt, and Y. Shavitt, "A QoS-aware multicast routing protocol," in *Proc. IEEE INFOCOM*, 2000.
[16] S. Chen, K. Nahrstedt, and Y. Shavitt, "A QoS-aware multicast routing protocol," *IEEE J. Sel. Areas Commun.*, vol. 18, no. 12, Dec. 2000.
[17] K. Calberg and J. Crowcorft, "Building shared tress using a one-to-many joining mechanism," *ACM Computer Commun. Rev.*, vol. 27, no. 1, pp. 5–11, Jan. 1997.
[18] M. Faloutsos, A. Banerjea, and R. Pankaj, "QoSMIC: Quality of service multicast internet protocol," *ACM SIGCOMM*, Aug. 1998.
[19] S. Yan, M. Faloutsos, and A. Banerjea, "QoS-aware multicast routing for the internet: The design and evaluation of QoSMIC," *IEEE/ACM Trans. Netw.*, vol. 10, no. 1, Feb. 2002.
[20] T. Ballardie, "Core based tree (CBT) multicast-architecture overview and specification," IETF Draft, 1995.
[21] S. Deering, D. L. Estrin, D. Farinacci, C. Liu, V. Jacobson, and L. Wei, "The PIM architecture for wide-area multicast routing," *IEEE/ACM Trans. Netw.*, vol. 4, no. 2, 1996.
[22] H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithm*, MIT Press, 1990.
[23] SteinLib Testdata Library. [Online]. Available: http://elib.zib.de/steinlib/steinlib.php

[24] B. M. Waxman, "Routing of multipoint connections," *IEEE J. Sel. Areas Commun.*, vol. 6, no. 9, Dec. 1988.



Stavros Vrontis was born in Athens, Greece, in 1975. He received his Diploma in Electrical and Computer Engineering from the National Technical University of Athens (NTUA) in 1998. He received his Ph.D. in the area of telecommunications and computer networks from NTUA in 2005. He had been working with the NTUA Telecommunications Laboratory as a research associate. He has been actively involved in several European research projects in the field of computer networks. Since 2005, he has been working for Vodafone Greece. His research work lies in the areas of quality of service, multicasting, active networks, mobile networks, network optimization, and distributed architectures.

Stavros Xynogalas was born in Athens, Greece, in 1975. He received his Diploma in Electrical and Computer Engineering from the National Technical University of Athens (NTUA) in 1997. He received his Ph.D. in the area of telecommunications and computer networks from NTUA in 2005. He has been actively involved in several research projects in the field of computer networks and telecommunications services, his current scientific fields of interest. His research work lies in the areas of pervasive computing, context-aware systems, network optimization, distributed architectures, intelligent systems, and mobile service engineering.

Efstathios Sykas was born in Athens, Greece, in 1956. He received the Dipl.-Ing. and Dr.-Ing. degrees in Electrical Engineering both from the National Technical University of Athens, Greece, in 1979 and 1984, respectively. From 1979–1984, he was a teaching assistant, during 1986–1988, he was a lecturer, then an assistant professor, 1988–1992, associate professor, 1992–1996 and now professor in the Division of Communications, Electronics and Information Engineering, Department of Electrical Engineering, National Technical University of Athens (NTUA). He has served as director of the Computer Science Division 1997–2000 and since 1999 is a member of the board of Governors of ICCS. He is a reviewer for several international journals and a member of IEEE Standard 802 committee. Dr. Sykas is a member of IEEE, ACM, and the Technical Chamber of Greece.