A Unicast-based Approach for Streaming Multicast

Reuven Cohen and Gideon Kaempfer
Department of Computer Science
Technion, Haifa 32000, Israel

Abstract—**Network layer multicast is know as the most efficient way to support multicast sessions. However, for security, QoS and other considerations, most of the real-time application protocols can be better served by upper layer (transport or application) multicast. In this paper we propose a scheme called M-RTP for multicast RTP sessions. The idea behind this scheme is to set up the multicast RTP session over a set of unicast RTP sessions, established between the various participants (source and destinations) of the multicast session. We then address the issue of finding a set of paths with maximum bottleneck for an M-RTP session. We show that this problem is NP-Complete, and propose several heuristics to solve it.**

I. INTRODUCTION

Over the past few years the Internet has seen a rise in the number of new applications that rely on multicast transmission. In these applications, one sender sends data to a group of receivers simultaneously. In a network that supports IP multicast, the sender needs to send only a single packet, which is then replicated by fork routers in the multicast delivery tree. Many protocols for supporting Network layer multicast have been developed, like DVMRP [7], CBT [4], and PIM [8] for Intra-AS multicast and BGMP [3] for Inter-AS multicast.

However, multicast can be supported not only in the network layer, but also in the transport layer or in the application layer. In such a case the replication of data is performed by the upper layer, while using a unicast routing service from the network layer. A straightforward advantage of such approach is that multicast applications can be executed in networks that do not support IP layer multicast. However, this approach is useful for other reasons also in cases where multicast is supported in the IP layer. As an example, the approach presented in [2] can be viewed as supporting multicast in the UDP layer. The main benefit there is reducing the overhead associated with setting up and maintaining a multicast tree in the network layer. Other reasons for supporting multicast using unicast in the Network layer is to allow firewalls to handle better the traffic created by the multicast application, and to facilitate the allocation of UDP port numbers (with unicast, each participant is free to select its port number for each session, whereas for multicast a global agreement is needed).

However, the most important advantage is Quality of Service (QoS): the provisioning, maintenance and tracking of QoS over a unicast path is known to be a much easier problem than over a multicast tree. Therefore, by provisioning upper layer multicast over a set of unicast Network layer paths it becomes much easier to guarantee QoS for a multicast session.

The theoretical problems, the main concepts and the algorithms presented in this paper are general. They can therefore be employed for addressing multicast-related issues in different contexts and under different assumptions. However, we selected to present them in the specific context of streaming multicast sessions. We consider an RTP multicast session, with a source S that distributes real-time traffic to a group of destination nodes. Providing QoS over a set of unicast paths is feasible today using the mechanisms and tools provided by frameworks like MPLS and DiffServ. However, the provisioning of QoS in a DiffServ network for an IP multicast routing tree is a tough problem which has not been addressed yet.

Moreover, RTP has a lightweight companion protocol called RTCP (Real Time Control Protocol), whose main purpose is to monitor the QoS of the RTP packets. RTCP is based on the periodic transmission of control messages to all participants in the session, using the same distribution mechanism as the data packets [14]. RTP receivers provide reception quality feedback using RTCP reports. An RTCP report contains information on the highest sequence number received, the number of packets lost, a measure of the interarrival jitter and timestamps needed to compute an estimate of the round-trip delay between the sender and the receiver issuing the report. These reports can be used by the sender in order to estimate the QoS perceived by the destinations and to adjust its transmission accordingly. For instance, when too many packets are lost, the sender can switch to a more aggressive compression scheme, or require the network to allocate more bandwidth. However, when multicast is performed in the IP layer, analysis the RTCP reports becomes a very difficult task. As an example, Figure 1 shows a multicast session, whose destination group consists of 2 hosts: d_1 and d_2, assuming that IP multicast is employed. Suppose that the sender s is informed that only 90% of the packets are received by d_1, and only 95% are received by d_2. Even if s knows the exact structure of the tree, it cannot determine how much of the loss should be attributed to each of the 3 paths comprising the tree: $s \rightsquigarrow x$, $x \rightsquigarrow d_1$ and $x \rightsquigarrow d_2$. For instance, two possible loss patterns are: (a) no loss on $s \rightsquigarrow x$, 10% loss on $x \rightsquigarrow d_1$ and 5% loss on $x \rightsquigarrow d_2$; (b) 5% loss on $s \rightsquigarrow x$, 5% loss on $x \rightsquigarrow d_1$ and no loss on $x \rightsquigarrow d_2$. Between these two extreme patterns, there exist an infinite number of additional different possible patterns. Therefore, there is no efficient method to improve the QoS in this case. This problem is also recognized in [1], where a mechanism that employs special probes from the sender to the receivers is proposed.

A trivial way to support a multicast application over a unicast routing layer is to set up a different unicast session between the host and every destination. However, this approach overloads the source and substantially increases the required bandwidth compared to the case where IP layer multicast is used. Another approach would be to use the fork nodes of the multicast tree (e.g. node x in Figure 1) as "active nodes" that participate in the higher layer protocol. For instance, if node x participates in the RTP layer of the session considered above, 3 RTP sessions are defined: between s and x, between s and d_1, and between s and d_2. Since each RTP session is associated with its

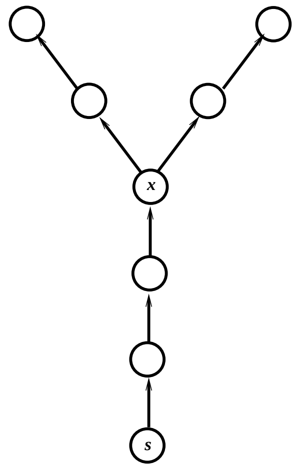

Fig. 1. A simple multicast session

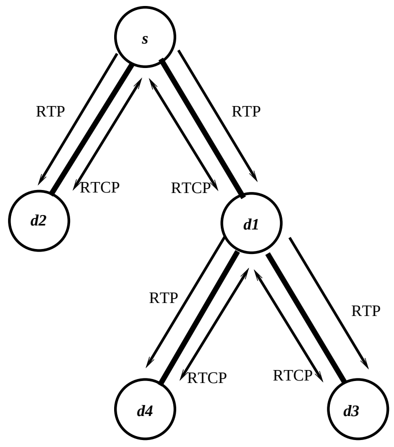

Fig. 2. Example of M-RTP

own RTCP, each of the 3 senders can easily analyze the RTCP reports it receives, in order to maintain the QoS of the stream it generates. This approach combines the advantage of IP layer multicast, namely scalability and efficiency, with the advantages mentioned above of upper layer multicast. However, this approach requires the the network routers to participate in higher layer protocols, in contrast to the fundamental design principal of the Internet: "put smarts in the ends of the network hosts, leaving the core dumb".

In this paper we study a third approach, that can be viewed as a combination of the two proposed above. In this approach, the multicast tree consists of multiple unicast paths. However, only members of the multicast session – namely the source s and the destination nodes – can sit on the end points of the paths. The network routers are assumed to perform simple unicast routing only. This idea was proposed in the first time in [2], in the context of increasing the scalability of multicast tree maintenance. However, in this paper we concentrate on the establishment of such a tree while *fulfilling the bandwidth requirements of a real-time application*. We show that the problem of finding such a tree with sufficient bandwidth is NP-Complete. We then propose heuristic solutions and analyze their performance.

The rest of the paper is organized as follows. Section II presents the main concepts of the considered scheme called M-RTP. In Section III the routing problem associated with M-RTP is studied. It is shown that finding an "M-RTP tree", namely a collection of paths that reaches each destination node, while guaranteeing the required bandwidth for the application is an NP-Complete problem. In Section IV we present two heuristics for solving this problem and in Section V we analyze the performance of these algorithms. Finally, section VI concludes the paper.

II. The M-RTP Approach for a Multicast RTP Session

The main idea behind M-RTP is to replace a single multicast RTP session with multiple unicast RTP sessions. In M-RTP each multicast group member is involved in one RTP session as a receiver, and in 0 or more RTP sessions as a proxy RTP sender. As an example, consider a multicast RTP session where s is the source and $M = \{d_1, d_2, d_3, d_4\}$ is the destination group. Instead of establishing a single RTP session, an M-RTP-based solution is presented in Figure 2. In this solution node s has a unicast RTP session with d_1 and another unicast RTP session with d_2. In these 2 sessions, s serves as the RTP source, while d_1 and d_2 serve as two independent RTP receivers. Host d_1 participates in 2 additional unicast RTP sessions, but this time as a proxy RTP sender[1]: with d_3 and d_4.

M-RTP restricts the replication of data *only to the parties actively participating in the multicast session*. As already indicated, this restriction bears several major advantages. The first advantage is that this approach requires no multicast support from the underlying routing layer. The second advantage is that QoS provisioning becomes a much easier task. And finally, the maintenance of QoS is significantly streamlined due to the following two reasons. Firstly, the reports received by every sender or receiver are related to a path rather than to a tree. Secondly, due to the RTCP bandwidth scaling algorithm [14], the number of reports received by every sender or receiver is $|M|$ times larger than in the case of a multicast tree though the total bandwidth consumed by these reports is equal in both cases.

A possible disadvantage of M-RTP is the cost of the routing. For instance, in certain cases, M-RTP causes multiple copies of the same application data to be transmitted over a single link. For instance, Figure 3(a) presents an undirected graph over which a multicast session from s to $M = \{d_1, d_2\}$ has to be established. Figure 3(b) presents a solution where a single multicast RTP session is established over a multicast tree. Figure 3(c) presents an M-RTP solution where the multicast session is carried out using 2 unicast RTP sessions: from s to d_1 and from s to d_2. Whereas in Figure 3(b) each packet of the multicast session is sent over the link $s - x$ only once, such a packet is sent twice over the same link in Figure 3(c). Although worst case analysis may show this to be a major problem, simulations suggest that the effect of M-RTP on routing cost is minimal. Unlike RTP mixers and translator [14], M-RTP does not impose processing of application data at the end point of an RTP con-

[1] In the terminology of RTP as defined in [14], a proxy-RTP sender can be viewed as a simple "RTP translator" or "RTP mixer".

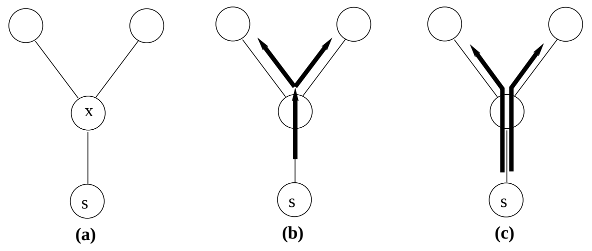

Fig. 3. Inefficient routing for M-RTP.

nections. Hence, the only delay an RTP message encounters due to M-RTP is the short time required in order to forward the message from one RTP connection to another, while updating the various RTP header fields.

The idea of restricting the multicast tree to contain fork nodes only from the group of end nodes was suggested in the first time in [2]. However, the algorithms presented there aimed at finding a collection of unicast routes whose overall cost is minimized. In contrast, the algorithms presented in this paper, in the context of M-RTP, aim at finding a collection of unicast routes that have sufficient bandwidth to support an RTP application. Reference [6]

III. The Problem of Routing for M-RTP

A. Introduction

Many multicast routing algorithms have been proposed in the literature. Most of these seek to optimize the *cost* of the multicast tree that is normally defined as the sum of costs associated with the communication links used by the tree. Preferring low cost trees may be a practical model for some networks, but many other models seem just as practical. For instance, a *minimum height* tree is relevant for delay sensitive applications, and a *maximum bottleneck* tree is relevant for bandwidth sensitive applications. The actual bandwidth a multicast application can utilize is no more than the bandwidth of the smallest bandwidth link, or *bottleneck*, in the multicast tree. This is most accurate in the case of real-time multicast applications where all the destinations must receive the same data from the source. For this reason, although a routing algorithm could construct a low total cost multicast tree, the low utilization of this tree, due to a local bottleneck, could render the tree much more "expensive" than intended. Therefore, *maximum bottleneck trees* may be of great practical importance.

The communications network is often modeled as an undirected graph. This implies that only one weight is assigned to an edge, regardless of the direction in which it is used. This implication is in many cases unrealistic and especially in the case where the weight associated to an edge represents its available bandwidth. For instance, consider a heavily loaded video server connected to a network by a single link. Most of the traffic going through this link would be from the server to the network, while the incoming traffic to the server, containing mainly RTSP requests and RTCP reports, would be only a small fraction of the total traffic. Thus, even if the physical bandwidth available in both directions of a link is equal, the available bandwidth on the link connecting the server would probably be distributed in

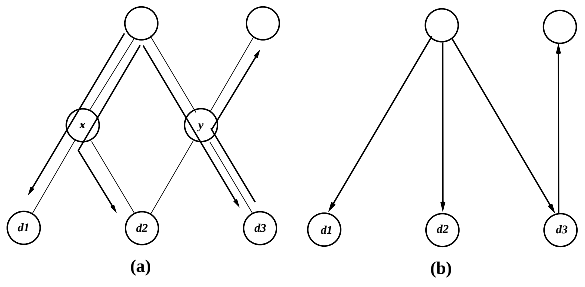

Fig. 4. An M-RTP session (a), and the "virtual multicast session tree" with which it can be associated

a highly asymmetrical way. This is even more likely to occur if the physical bandwidth is not symmetrically distributed, as for ADSL, cable-modem or satellite links.

B. Graph theoretical problems related to routing for M-RTP

Figure 4 shows an example for an M-RTP session, and the associated session tree. As already explained, M-RTP uses a set of unicast sessions, that can be viewed as a "virtual multicast session tree" Each fork node in this tree must be either the source of the multicast RTP session, or a member of the destination group.

The major restriction on the structure of the underlying session tree is that every node with an outgoing degree of more than 1 in the tree *must* be either the source or a member in the multicast group M (see Figure 4(b)). This restriction will be referred to as the *fork restriction*. The formal definition of a new set of optimization problems related to M-RTP is given below:

Problem 1: λ-Optimal Multicast Path Set Problem (MPSP(λ)). Given a graph $G(V, E)$, a weight function $w: E \rightarrow \mathcal{R}^+$, a group of "active nodes" $A \subseteq V$, a multicast group $M \subseteq A$ and a multicast source $s \in M$, find a set of paths T that satisfies a given optimality criterion λ and:
1. $\forall p \in T$ the endpoints of p are vertices in A.
2. $\forall v \in M$ there exists a path $p^v = s \rightsquigarrow v$ that is a concatenation of a subset of paths from T.

The solution to MPSP(λ) induces a *"virtual multicast session tree"* spanning M. This tree is a "real" spanning tree in an induced graph $G'(A, E')$ where an edge between two vertices in A exists if and only if a path between these vertices exists in the original graph G. Note, that a solution to MPSP(λ) may actually induce a subgraph that is *not* a "real" tree in G. For instance, the multicast session tree depicted in Figure 4(b) is not a tree in the original graph shown in Figure 4(c) because it contains a directed cycle between d_3 and y.

When $A \equiv V$, MPSP(λ) reduces to the problem of finding a "real" tree rooted at s and spanning M while satisfying λ. However, in this paper we consider the version of this problem where $A \equiv M$, namely every end-point of the paths in the set T is either the source s or belonging to the multicast group M. An interesting generalization is when A contains nodes in $V - M$. Namely, allowing some of the network routers to have the capability of serving as end points of RTP sessions.

The basic optimization criterion usually defined for multicast trees is the multicast tree cost. The problem relevant to M-RTP is defined as follows:

Problem 2: Minimum Sum Multicast Path Set Problem (MPSP(MinSum)).
Find a solution T to MPSP(λ) where λ is defined as minimizing: $\sum_{p \in T} \sum_{e \in p} w(e)$.

This definition penalizes a solution for using an edge several times by multiplying its weight by the number of times it is used. For example, consider again Figure 4(a), and assume that all edges in the graph have an equal cost of 1. The depicted M-RTP session has a cost of 8, since the edge $s \rightarrow x$ is used twice and the edge $d_3 \rightarrow y$ is used in both directions. If x had been in group A, we could have an RTP session between s and x, a session between x and d_1, and a third session between s and d_2. The cost of the resulting session would have been reduced from 8 to 7. From the same considerations, the cost could have been reduced by one unit if node y had been in A.

For the case $A \equiv V$, MPSP(MinSum) reduces to the classic Steiner Tree Problem (STP) (see [10]). Several variations on STP have been defined in the literature. Two such variations are the Directed STP [13], defined like STP but specifically for directed graphs, and the Dynamic STP [11], defined for dynamically changing multicast groups.

In [13], the SCTF algorithm for the construction of directed Steiner Trees is presented. Applying a minor revision to this algorithm results in an algorithm for the directed MPSP(MinSum) which achieves the exact same approximation ratio that SCTF achieves for STP. Static and dynamic approximation algorithms for the undirected MPSP(MinSum) and its dynamic counterpart are presented in [2]. These algorithms achieve approximation ratios that are asymptotically equivalent to the ratios achieved for the non-restricted problems.

However, in the context of real-time RTP sessions, the most important criterion is the availability of bandwidth along the path of every session. Let β be the bandwidth of the data generated by the source. Assuming that the RTP nodes do not change the coding and the bandwidth, a legal multicast session tree is one that consists of unicast paths whose available bandwidth is $\geq \beta$. To find such a tree, we are seeking for the *maximum bottleneck multicast tree.* If this tree has a bandwidth larger than β, then it can be used for the considered multicast session. If the maximum bottleneck multicast session tree has a bandwidth smaller than β, then there is no possible way to establish the considered RTP session in the network without using multicast within the network. The maximum bottleneck version of the multicast path set problem is defined as follows:

Problem 3: Find a solution T to MPSP(λ) where λ is defined as maximizing the minimal bandwidth allocated to a path in T. More formally, λ is defined as maximizing: $\min_{\{e|e \in p, p \in T\}} \frac{w(e)}{|\{p|p \in T, e \in p\}|}$, where $w(e)$ is the bandwidth available on edge e for the multicast session.
For example, in Figure 4, assume that all edges in the graph have an equal capacity of 1 on every direction. The routing depicted has two bottleneck paths, $s \rightsquigarrow d_1$ and $s \rightsquigarrow d_2$ of capacity $\frac{1}{2}$, that share a common edge $s \rightarrow x$. Therefore, the bandwidth available for this tree is $1/2$. If we allowed the network nodes to support multicast, namely node x could receive a single packet from s and sends one copy to d_1 and another to d_2, then we could serve a multicast session of 1 rather than $1/2$.

Solving the *non-restricted* problem (namely the case where $A \equiv V$) for the maximal bottleneck optimization criteria, i.e. finding a tree spanning the multicast group with a maximal value for $\min_{e \in T}\{w(e)\}$, is considered easy. This is because many polynomial time algorithms solve for this problem. For instance, the SMMT algorithm in [5] solves this problem if it is used for weights $\frac{1}{w(e)}$. Another option is to run a variation on the Prim algorithm, as shown in Section IV-C. However, MPSP(λ) is NP-Hard for many natural criteria λ, including the maximal bottleneck criterion.

C. On the Hardness of MPSP(MaxBottleneck)

M-RTP introduces a new perspective to multicast routing problems. It restricts the multicast fork nodes of the virtual multicast session tree to the multicast group, thus introducing a basic constraint that must be satisfied even before optimization criteria can be approached. Clearly, this additional constraint leaves previously hard optimization problems, such as STP, in the NP-Hard domain [10]. The question is whether other optimization problems, such as the maximal bottleneck multicast tree, become any harder.

In Section III-B a formal definition of optimization problems subject to the fork restriction imposed by M-RTP was introduced. In what follows we show that even without the optimization criterion λ of MPSP(λ), *if every edge may be used at most once*, finding a fork restricted routing is NP-Complete. From this result follows that MPSP(MaxBottleneck) is NP-Hard[2].

In order to satisfy the fork restriction imposed by M-RTP, it is sufficient to find a set of paths inducing a virtual tree spanning the multicast group with fork nodes exclusively within this group. We define the edge disjoint multicast path set problem (MPSP^E) as follows:

Problem 4: Edge Disjoint Multicast Path Set Problem (MPSP^E).
Given a graph $G(V, E)$, a group of active nodes $A \subseteq V$, a multicast group $M \subseteq A$ and a multicast source $s \in M$, find a set T of *edge disjoint*, but not necessarily vertex disjoint, paths that satisfies:
1. $\forall p \in T$ the endpoints of p are vertices in A.
2. $\forall v \in M$ there exists a path $p^v = s \rightsquigarrow v$ that is a concatenation of a subset of paths from T.

The "virtual multicast session tree" shown in Figure 4(a) is not a valid solution for MPSP^E because the link $x \rightarrow s$ is used twice. However, a solution to MPSP^E defines a virtual tree spanning M but may still induce a subgraph that is still *not* a tree. For instance, consider Figure 4(a) again, and suppose that M consists of only d_3 and d_4. Then, the set of paths $\{s \rightarrow y \rightarrow d_3, d_3 \rightarrow y \rightarrow d_4\}$ is a valid solution for MPSP^E that is not a "real" tree because of the directed cycle between d_3 and y. MPSP^E is closely related to MPSP(λ) except that the optimization criterion is replaced by the restriction that the paths in the solution remain edge disjoint. In what follows, it will be shown that MPSP^E is NP-Complete. The consequence of this result is that many natural optimization problems stemming from MPSP(λ) are NP-Hard too, since MPSP^E can be reduced to them. The hardness of MPSP^E will be proven by reducing the

[2] These results pertain for directed graphs. For undirected graphs we have only partial evidence for the hardness of MPSP(λ).

Directed Hamiltonian Circuit (DHC) problem [10] to it. DHC is defined as follows:

Definition 1: Directed Hamiltonian Circuit (DHC).
Given a directed graph $G(V, E)$, does G contain a directed Hamiltonian circuit (i.e. a simple circuit that passes through all vertices in V) ?
The following theorem proves the hardness of MPSP^E.

Theorem 1: MPSP^E on directed graphs is NP-Complete

Proof: We show a reduction from DHC:
Given a directed graph $G(V, E)$, construct the following directed graph $G'(V', E')$ (see Figure 5). First, choose an arbitrary source vertex $s \in V$.

$$\begin{aligned} V' &\triangleq \left\{v^{in}, v^{mid}, v^{out} | v \in V \backslash \{s\}\right\} \cup \\ &\quad \left\{s^{in}, s^{mid1}, s^{mid2}, s^{out}\right\} \\ E' &\triangleq \left\{(v^{in}, v^{mid}), (v^{mid}, v^{out}) | v \in V \backslash \{s\}\right\} \cup \\ &\quad \left\{(s^{in}, s^{mid1}), (s^{mid1}, s^{mid2}), (s^{mid2}, s^{out})\right\} \cup \\ &\quad \left\{(v^{out}, w^{in}) | (v, w) \in E\right\}. \end{aligned}$$

Define a multicast group $M' \triangleq \left\{v^{mid} | v \in V\right\} \cup \left\{s^{mid1}, s^{mid2}\right\}$ and a multicast source $s' \triangleq s^{mid2}$ in G'. It is now shown that a solution to DHC on G exists if and only if a solution to MPSP^E on G' exists assuming $A' \equiv M'$. A directed Hamiltonian circuit C in G can be transformed into a directed Hamiltonian path P' in G' as follows. For every edge $(u, v) \in C$ add the edge (u^{out}, v^{in}) to P'. For every vertex $v' \in V' \backslash s'$ add the edges $\left\{(v^{in}, v^{mid}), (v^{mid}, v^{out})\right\}$ to P'. Finally, add the edges $\left\{(s^{in}, s^{mid1}), (s^{mid2}, s^{out})\right\}$ to P'. Since P' touches all vertices in $V' \backslash \left\{s^{mid1}, s^{mid2}\right\}$ exactly twice and touches $\left\{s^{mid1}, s^{mid2}\right\}$ exactly once, it must be a directed Hamiltonian path in G' starting at s'. Now, P' can be cut into a set T' of subpaths connecting vertices in M', thus creating a valid solution to MPSP^E on G'. To prove the other direction, let T' be a solution to MPSP^E on G'. T' is by definition a set of *edge disjoint* paths, and by construction, for every vertex $v' \in V'$, its in-degree or its out-degree (or both) is exactly 1. Therefore, T' must be a set of paths that form a single simple path P' when united. This path must start at s^{mid2}, traverse every vertex in M' and finally reach s^{mid1}. Thus, it induces a directed Hamiltonian circuit in G. Clearly, MPSP^E $\in$ NP, and thus MPSP^E is NP-complete. ■

MPSP^E holds the essence of the fork restriction imposed by M-RTP. Therefore, it can be expected that any optimization problem based on MPSP^E is likely to be NP-Hard. Specifically, MPSP(MaxBottleneck) is such a case.

Theorem 2: MPSP(MaxBottleneck) is NP-Hard.

Proof: The following is a reduction from MPSP^E to MPSP(MaxBottleneck). Given an input tuple (G, A, M, s) for MPSP^E, we create an input tuple (G, A, M, s, w') for MPSP(MaxBottleneck) where $w'(e) \equiv 1$. If the bottleneck of an optimal solution T^* for MPSP(MaxBottleneck) is 1, it can be deduced that every edge in T^* is used exactly once. Thus, T^* is a solution to MPSP^E too. On the other hand, if the bottleneck of T^* is less than 1, no solution to MPSP(MaxBottleneck) exists that uses every edge at most once. Hence, no solution to MPSP^E exists. ■

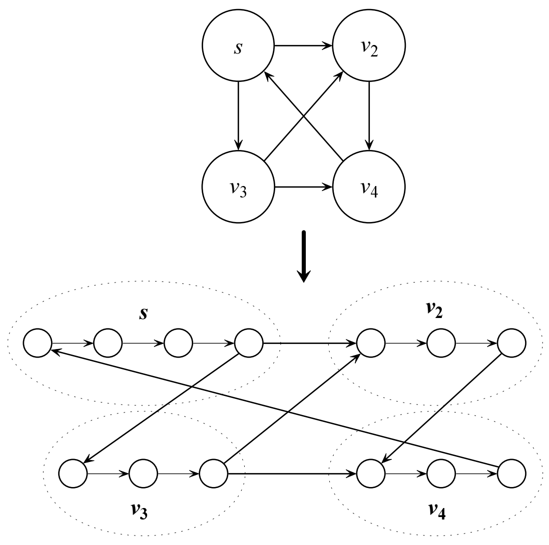

Fig. 5. Construction example for the reduction from DHC to MPSP^E.

IV. Maximum Bottleneck Routing Algorithms for M-RTP

In this section we propose two algorithms for approximating MPSP(MaxBottleneck). In what follows the approximation ratio of an algorithm is defined as follows:

Definition 2: Approximation ratio of algorithm A.
For every instance p of a problem P, let $B(p)$ be the value of the solution found by A and $B^*(p)$ the value of the optimal solution. The approximation ratio of algorithm A is either:

$$\min_{p \in P} \left\{ \frac{B(p)}{B^*(p)} \right\}$$

for maximization problems, or:

$$\max_{p \in P} \left\{ \frac{B(p)}{B^*(p)} \right\}$$

for minimization problems.

The first algorithm is referred to as the Widest Path Heuristic (WPH). It achieves a solution that is no worse than $\frac{1}{O(|M|)}$ times the bottleneck size of the optimal solution, where $|M|$ is the size of the multicast group. Although this approximation ratio can be achieved by other algorithms, simulations show that WPH performs nearly optimally (see Section V).

The second algorithm, referred to as the Double Tree Heuristic (DTH), achieves an approximation ratio that is dependent on the graph asymmetry rather than the size of the multicast group. DTH assumes that every directed edge $u \to v$ has an antisymmetric edge $v \to u$. In symmetrically loaded networks, the ratio guaranteed by DTH can be better than the ratio guaranteed by WPH, even if the multicast group is of small size. For undirected graphs, the approximation ratio DTH achieves is exactly $\frac{1}{2}$.

A. The Widest Path Heuristic (WPH)

WPH builds a virtual tree spanning M. The algorithm starts with the source vertex s. In each iteration of the algorithm, a maximum *residual* bottleneck path is considered from every covered vertex in M to every vertex in M that has not yet been covered. The *residual* weight of an edge e used n times by the algorithm is defined as $\frac{w(e)}{n+1}$, where $w(e)$ is the available bandwidth of the edge. The path with the maximal bottleneck is added to the solution. Thus, after each iteration, at least one more vertex from M is covered. As shown later, this algorithm achieves an approximation ratio of $\frac{1}{O(|M|)}$ in the worst case. A formal definition of the algorithm is given below:

Algorithm 1: Widest Path Heuristic (WPH).

```
1. Initialization: $T \leftarrow \Phi, X \leftarrow \{s\}$.
2. While $M \backslash X \neq \Phi$ do
   (a) Find the maximal residual bottleneck path $p$ from $X$ to a vertex $m$ in $M \backslash X$.
   (b) $X \leftarrow X \cup m$.
   (c) Update all residual weights of edges in $p$ (i.e the new residual weight of an edge $e$ used $n$ times so far is $\frac{w(e)}{n+1}$).
   (d) $T \leftarrow T \cup p$.
3. T is the desired solution.
```

The implementation of finding the maximal residual bottleneck path in step (2a) is similar to the construction of a maximal bottleneck tree. A variation of the Prim algorithm for finding a minimum spanning tree [12] can be used as follows. Two sets of vertices are used: a set of "known" vertices and a set of "unknown" vertices. In each iteration, the unknown vertex connected by the edge of greatest capacity to a known vertex is added to the known set. The algorithm terminates when all unknown vertices become known. If the initial set of known vertices is a single vertex s, the algorithm constructs a maximal bottleneck spanning tree rooted at s (see Section IV-C). We use this algorithm slightly differently for the implementation of step (2a). The initial set of known vertices is the set of vertices X already covered by the algorithm. The algorithm stops as soon as an unknown vertex of M becomes known. Every time a vertex is added to the known set, the edge through which it is discovered is remembered. At the end of the algorithm, a maximal bottleneck path from a vertex in the initial known set can be reconstructed by backtracking these stored edges from any initially unknown vertex. In this manner, a minimal bottleneck path is found at the end of the Prim algorithm variant. For a formal proof, see Section IV-C. The algorithm can be implemented in $O(|E| + |V| \log |V|)$ time using a Fibonacci heap [9]. Hence, the total time complexity of WPH is no more than $O(|M|(|E| + |V| \log |V|))$.

Lemma 1: WPH achieves an approximation ratio of $\frac{1}{O(|M|)}$.

Proof: Denote by $B(X)$ the bottleneck of a solution X for MPSP(MaxBottleneck), and by $B(p)$ the residual bottleneck of a path p. Let T^* be an optimal solution, and T a solution found by WPH. Let $p = x \rightsquigarrow y$ be the last path added to T with $B(p) = B(T)$. According to step (2a) of the algorithm, when p is added to T it has a larger bottleneck than any other path p' connecting a vertex from the multicast group already covered by WPH with a vertex from the multicast group yet to be covered, namely:

$$\forall p' \in \{x' \rightsquigarrow y' | x' \in X, y' \in M \backslash X\}, B(p) \geq B(p'). \qquad (1)$$

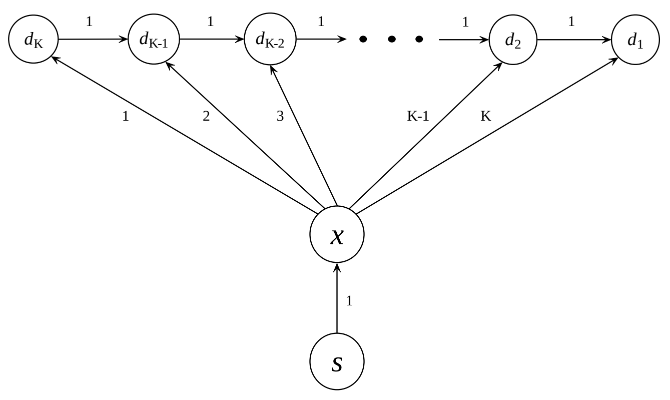

Fig. 6. A worst case network for WPH

Since $s \in X$ at all stages of the algorithm, (1) implies that:

$$\forall p' = s \rightsquigarrow y, \ where \ y \in M \backslash X, B(p) \geq B(p'). \qquad (2)$$

Let p^* be a simple path embedded over the path connecting s to y in the optimal solution T^* (e.g. in Figure 4(a), the path from s to d_4 is $s \rightarrow y \rightarrow d_3 \rightarrow y \rightarrow d_4$ whereas a simple path embedded on this path is $s \rightarrow y \rightarrow d_4$). By (2), at the time p is selected, $B(p) \geq B(p^*)$ holds. Hence, we have:

$$B(T) = B(p) \geq B(p^*). \qquad (3)$$

Let e^* be an edge in p^* with the minimal weight. Every edge can be used no more than $|M| - 1$ times during the algorithm, since all maximal bottleneck paths (found in step (2a)) are simple paths, and no more than $|M| - 1$ iterations of the *while* loop are performed. Hence:

$$B(p^*) \geq \frac{w(e^*)}{|M| - 1} \geq \frac{B(T^*)}{|M| - 1}. \qquad (4)$$

Thus, by (3) and (4), $B(T) \geq \frac{B(T^*)}{O(|M|)}$. ■

The ratio of $\frac{1}{O(|M|)}$ is a strict bound as demonstrated in the following example. Consider the directed graph in Figure 6. missing antisymmetric edges may be defined as having a weight of less than $\frac{1}{|M|}$. For simplicity of presentation, these edges are ignored. Clearly, an optimal solution for MPSP(MaxBottleneck) on the above defined network, must have a bottleneck of 1 at the most, and such a solution indeed exists: $T^* \triangleq \{s \rightarrow x \rightarrow d_K\} \cup \{d_i \rightarrow d_{i-1} | 2 \leq i \leq K\}$. However, WPH may cover vertex d_i in the i^{th} iteration and the solution it finds is: $T = \{s \rightarrow x \rightarrow d_i | 1 \leq i \leq K\}$. This solution has a bottleneck of $\frac{1}{|M|-1}$ since the edge (s, x) is used K times. Thus, the approximation ratio achieved in this example is $\frac{1}{O(|M|)}$.

B. The Double Tree Heuristic (DTH)

DTH is aimed at achieving an approximation ratio that is independent of the size of the multicast group. The asymmetry of a graph G is defined as the maximal ratio between the weights of a pair of antisymmetric edges. Formally, the following notation proposed in [13] is used:

Definition 3: Maximum edge asymmetry.

Let $G = (V, E)$ be a directed graph with a weight function $w\colon E \to \mathcal{R}^+$. Define

$$\Psi_m(G) \quad \triangleq \quad \max_{(u,v)\in E} \frac{\max\{w(u,v), w(v,u)\}}{\min\{w(u,v), w(v,u)\}}$$

as the *maximum edge asymmetry*.
Unlike WPH, DTH assumes that every directed edge $u \to v$ has an antisymmetric edge $v \to u$. The algorithm has three phases. In the first phase, the weight of the bottleneck of a maximal bottleneck tree spanning M is computed. This can be done by constructing a maximal bottleneck tree, using the variation on Prim's MST algorithm presented in Section IV-C. In the example depicted in Figure 7, the bottleneck size is 6.

In the second phase, from all the trees whose bottleneck is equal to the maximal bottleneck, a tree with a maximal *reverse bottleneck* is found. The *reverse bottleneck* of an outgoing directed tree is defined as the minimum of the antisymmetric weights. Note that these trees need not satisfy the *fork restriction*. The second phase can again be implemented using a variation on the Prim algorithm. In the example, this tree (c) differs from the first tree constructed in (b) since the reverse bottleneck achieved is 3 (because of $d_1 \to x$) whereas the reverse bottleneck of the first tree constructed is 2 (because of $d_2 \to d_1$).

In the third phase, the tree found in the second phase is transformed into a set of paths with endpoints in the multicast group. The paths are constructed by touring the second tree in a depth-first manner. This tour is broken into subpaths such that every time a vertex belonging to the multicast group is passed, the previous subpath is ended and a new one is initiated. Finally, unneeded paths that lead to multicast destinations already reached by previous paths from our solution can be removed. In the example, the full tour includes three paths: $s \to x \to d_1$, $d_1 \to x \to d_2$ and $d_2 \to x \to s$. The last path returns to s and may therefore be omitted (d). The bottleneck of the solution found by DTH in the example is 3 (because of the edge $d_1 \to x$) whereas the optimal solution (e) achieves a bottleneck of 6. Note that any solution produced by DTH uses every edge once at the most.

The algorithm can be implemented to run in $O(|E| + |V|\log|V|)$ time.

Lemma 2: DTH achieves an approximation ratio of $\frac{1}{O(\Psi_m(G))}$.

Proof: Denote by $B(X)$ and $B^r(X)$ the bottleneck and the reverse bottleneck of X respectively. Denote by T^* an optimal solution to MPSP(MaxBottleneck), by T the solution found by DTH, and by T_2 the tree found in the second phase of DTH. DTH guarantees that:

$$B(T) \geq \min\{B(T_2), B^r(T_2)\}$$

Note that the bottleneck of an optimal *non-restricted* spanning tree is clearly greater or equal to the bottleneck of an optimal solution to MPSP(MaxBottleneck). Thus, and since T_2 is an optimal non-restricted spanning tree, if $B(T_2) \leq B^r(T_2)$ then $B(T) \geq B(T_2) \geq B(T^*)$, namely $B(T) = B(T^*)$.

If $B(T_2) > B^r(T_2)$, let $\tilde{e}^r$ be a reverse edge in T_2 satisfying $w(\tilde{e}^r) = B^r(T_2)$. Let $\tilde{e}$ be the antisymmetric partner of $\tilde{e}^r$. Since $w(\tilde{e}) \geq B(T_2) > B^r(T_2) = w(\tilde{e}^r)$, by Definition 3: $\Psi_m(T_2) \geq \frac{w(\tilde{e})}{w(\tilde{e}^r)}$. Therefore:

$$B^r(T_2) = w(\tilde{e}^r) \geq \frac{w(\tilde{e})}{\Psi_m(T_2)} \geq \frac{B(T_2)}{\Psi_m(T_2)}$$

Thus, $B^r(T_2) \geq \frac{B(T_2)}{\Psi_m(T_2)}$ holds, and

$$B(T) \geq B^r(T_2) \geq \frac{B(T_2)}{\Psi_m(T_2)} \geq \frac{B(T^*)}{\Psi_m(T_2)} \geq \frac{B(T^*)}{\Psi_m(G)}$$

Therefore, regardless of the relation between $B(T_2)$ and $B^r(T_2)$, it always holds that:

$$B(T) \geq \frac{B(T^*)}{\Psi_m(G)}$$

■

Although the worst case performance of DTH can be guaranteed by performing the third phase on *any* maximal bottleneck tree rooted at s and spanning M, and not necessarily the one that has the maximal reverse bottleneck, the second phase is performed in order to improve the average performance. This is clearly demonstrated in simulations (see Section V).

C. An Algorithm for Finding a Maximal Bottleneck Tree

In Section IV-A a variant of the Prim algorithm [12] was introduced. For a given graph $G(V, E)$, this algorithm builds a maximal bottleneck spanning tree T rooted at a given vertex s. A formalization of this algorithm and proof of its correctness is given hereafter.

Algorithm 2: Maximum Bottleneck Tree (MBT) (Prim variant).

```
1. $K \leftarrow \Phi, U \leftarrow V$
2. $predecessor[s] := NULL$
$bottleneck[s] := \infty$
$\forall v \in U \setminus \{s\}, bottleneck[v] := 0$
$Q \leftarrow \{s\}$
3. while $Q \neq \Phi$
  (a) Let $v$ be a vertex in $Q$ with the maximal $bottleneck[v]$
  (b) $K \leftarrow K \cup \{v\}, U \leftarrow U \setminus \{v\}$
  (c) for every edge $v \xrightarrow{e} u$ such that $u \in U$:
if $bottleneck[u] < \min\{bottleneck[v], w(e)\}$ then
  i. $bottleneck[u] := \min\{bottleneck[v], w(e)\}$
  ii. $predecessor[u] := v$
  iii. $Q \leftarrow Q \cup \{u\}$
  (d) $Q \leftarrow Q \setminus \{v\}$
```

This algorithm constructs a $predecessor$ array that defines a tree rooted at s. Clearly, if the graph is connected, this tree spans the entire graph $G(V, E)$. Otherwise, only the vertices found in K at the end of the algorithm are reached by this tree. In the following lemma it will be shown that this is a maximal bottleneck tree.

Lemma 3: MBT constructs a maximal bottleneck tree spanning K.

Proof: It suffices to show that for every vertex v, if a path from s to v containing only edges in $predecessor$ exists, it is a maximal bottleneck path connecting s to v with a bottleneck

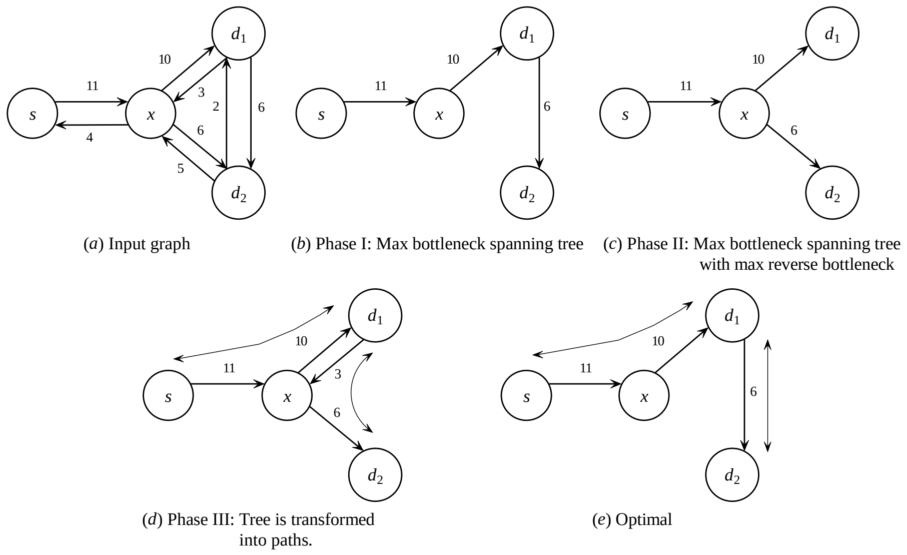
(a) Input graph
(b) Phase I: Max bottleneck spanning tree
(c) Phase II: Max bottleneck spanning tree with max reverse bottleneck
(d) Phase III: Tree is transformed into paths.
(e) Optimal
s
x
d1
d2
11
4
10
3
2
6
6
5

Fig. 7. The DTH algorithm

equal to $bottleneck[v]$. We prove by induction on the order by which vertices are added to the set of "known" vertices K. For s this is trivial. Assume the correctness of the induction assumption for all vertices in K at the beginning of step (3). We will prove the assumption for the vertex v added to K in step (3b).

Let $w \triangleq predecessor[v]$. By step (3c), w is already in K. Thus, by the assumption, $bottleneck[w]$ is the size of a maximal bottleneck path from s to w contained in $predecessor$. Also, by steps (3(c)i) and (3(c)ii), this path can be extended to v by the edge $predecessor[v]$ thus creating a path with a bottleneck of $bottleneck[v]$.

Assume towards a contradiction, that a maximal bottleneck path p from s to v has a bottleneck b that is greater than $bottleneck[v]$. p must pass from the set of "known" vertices K to the set of "unknown" vertices U by means of an edge $x \xrightarrow{e} y$, since $s \in K$ and $v \in U$. Since $b \leq w(e)$ and $x \in K$, by step (3(c)i):

$$bottleneck[y] \geq \min\{bottleneck[x], w(e)\} \geq \geq b > bottleneck[v].$$

Thus, by step (3a), y should have been added to K before v - a contradiction. ■

A variant of MBT can be used to construct a maximal bottleneck tree spanning only a subset X of the vertices in the graph. To achieve this, the algorithm must be stopped immediately after the last vertex in X has become "known". A formal proof of the correctness of this algorithm is similar to the proof for SMMT in [5].

Consider a minor change to MBT that allows more than one vertex to be initially put in the "known" set of vertices K: In the description of the algorithm, change every appearance of s into M, where M is a set of vertices. The resulting algorithm, finds a maximal bottleneck path from M to any vertex reachable from M. The proof for this algorithm can be constructed by changing every appearance of s into M in the above lemma.

V. Simulation Results

In the previous Section IV two approximation algorithms for MPSP(MaxBottleneck) were presented and analyzed for worst case performance. In this section we present the results these algorithms achieve on actual graphs that hopefully represent real-world communications networks. The main conclusion of these simulations is that the proposed algorithms, and in particular WPH, behave in practice almost optimally.

Two versions of DTH are considered. DTH is the version presented in the previous section, whereas DTH' is DTH without the second phase (i.e. the paths in the solution are created by running the third phase of the algorithm on an arbitrary maximal bottleneck tree). DTH' is presented only in order to assess the benefit of the second phase of DTH.

The algorithms were tested on 500 randomly generated networks, based on the model used in [2]. The weights (capacities) of the edges were initialized at random values uniformly distributed between 2 and 22. For every edge e with initial weight $w(e)$ used i times in the above paths, the edge weight was updated to be $\frac{w(e)}{i+1}$. On every graph, the algorithms were run on every multicast group size between 3 and 99 (this size includes the source node). The multicast groups were reselected at random for every graph and every group size.

Since MPSP(MaxBottleneck) is NP-Hard (see Section III-C), the optimal solution could not be computed. Instead, the bottleneck of a maximal bottleneck tree rooted at the multicast source and spanning the destinations was computed. This bottleneck is an upper bound on the value of the optimal solution since is disregards the fork restriction imposed by M-RTP. As shown in the following, it frequently equaled the optimal solution.

In Figure 8, the performance of the evaluated algorithms is presented as a function of the multicast group size. The most prominent property of the graph is that the optimal solution and the performance of the algorithms rapidly decline with the multicast group size. This behavior is expected since the greater the multicast group size, the more edges need to be used on the

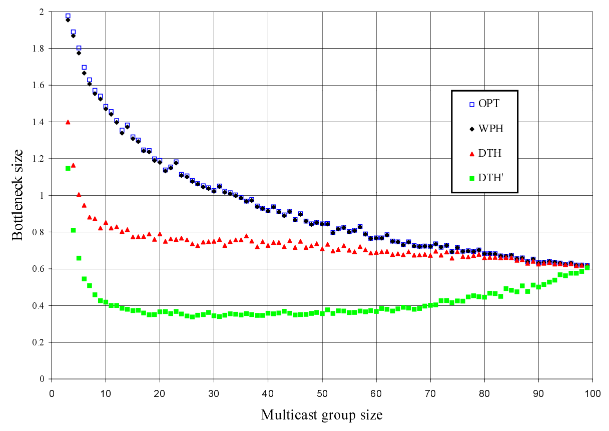

Fig. 8. Algorithm performance versus multicast group size

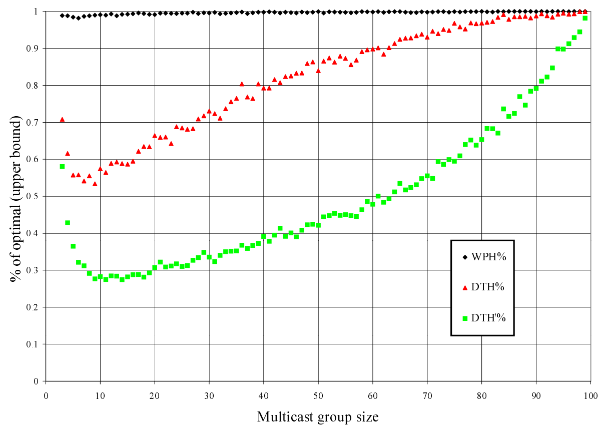

Fig. 9. Algorithm relative performance versus multicast group size

average, and thus the probability of using a low capacity edge increases.

In practice, WPH performs almost optimally, regardless of the multicast group size, despite the theoretical approximation ratio of $\frac{1}{O(|M|)}$. DTH performs relatively worse than WPH averaging 78% of the optimal solution. DTH' is consistently worse than DTH, averaging 45% of the optimal solution, thus justifying the second phase of DTH.

In Figure 9, the relative performance of the evaluated algorithms is presented as a function of the multicast group size. This graph highlights the results discussed previously.

Our main conclusion is that WPH is in practice almost optimal for any multicast group size. Although WPH is the only algorithm proposed that may cause the same data to be transmitted more than once over the same link, in practice this rarely occurs. For very large group sizes, DTH may be preferable, since it too achieves almost optimal solutions within less time.

Throughout our study we have also measured the cost penalty induced by the restriction that data is replicated only by the members of the multicast session. We have compared the cost of every tree to the cost incurred by the MBT algorithm, proposed in Section IV-C, which finds the maximal bottleneck tree without this restriction. It turns out that the average cost of the tree found by both WPH and DTH is only 5-15% higher than the average cost of the optimal tree found by MBT. These results are in accordance with those presented in [2].

VI. Conclusions

We have proposed a scheme called M-RTP for multicast RTP sessions. The main idea behind this scheme is to set up the multicast session over a set of unicast paths between the participants of the multicast session. The main advantage of this approach is the ease of QoS provisioning and maintenance. In particular, the information carried by RTCP reports can be more accurately analyzed in order to increase the allocated bandwidth over a path whose QoS falls short of expectations.

We have studied the problem of setting up a multicast tree under the constraints imposed by the new scheme. This gave rise to a new family of problems called MPSP (Minimum Path Set Problem). We have concentrated on the MPSP(MaxBottleneck) variation of this problem, which seeks for a set of paths that guarantees maximum bottleneck. This problem was shown to be NP-Complete and 2 heuristics were proposed: the Widest Path Heuristic (WPH) and the Double Tree heuristic (DTH). For each algorithm we analyzed the approximation ratio analytically, and the average performance by means of simulations. The simulations show that WPH is almost optimal for any group size.

References

[1] A. Adams, T. Bu, J. Horowitz, D. Towsley, R. Caceres, N. Duffield, F. LoPresti, S. Mon, and V. Paxson. The use of end-to-end multicast measurements for characterizing internal network behaviour. *IEEE Communications Magazine*, 38(5), 2000.

[2] E. Aharoni and R. Cohen. Restricted dynamic Steiner trees for scalable multicast in datagram networks. *IEEE/ACM Transactions on Networking*, 6(3), June 1998.

[3] A. Almeroth. The evolution of multicast: from the "mbone" to internet2 deployment. *IEEE Communications Magazine*, 14(1), 2000.

[4] T. Ballardie, P. Francis, and J. Crowcroft. Core based trees (CBT). In *SIGCOMM Symposium on Communications Architectures and Protocols*, pages 85–95, September 1993.

[5] Charles Chiang, Majid Sarrafzadeh, and C. K. Wong. Global routing based on Steiner min-max trees. *IEEE Transactions on Computer Aided Design*, 9(12):1318–1325, December 1990.

[6] Y. Chu, S. Rao, and H. Zhang. A case for end system multicast. In *ACM Sigmetrics*, Santa Clara, CA, June 2000.

[7] S. Deering, C. Partridge, and D. Waitzman. Distance vector multicast routing protocol. RFC-1075, November 1988.

[8] Stephen Deering, Deborah L. Estrin, Dino Farinacci, Van Jacobson, Liu Ching-Gung, and Liming Wei. The PIM architecture for wide-area multicast routing. *IEEE/ACM Transactions on Networking*, 4(2):153–162, April 1996.

[9] Michael L. Fredman and Robert Endre Tarjan. Fibonacci heaps and their uses in improved networks optimization algorithms. *Journal of the ACM*, 34(3):596–615, July 1987.

[10] Michael R. Garey and David S. Johnson. *Computers and Intractability - A Guide to the Theory of NP-Completeness*. W. H. Freeman and company, 1979.

[11] Makoto Imase and Bernard M. Waxman. Dynamic Steiner tree problem. *SIAM Journal on Discrete Mathematics*, 4(3):369–384, August 1991.

[12] R.C. Prim. Shortest connection networks and some generalizations. *Bell System Technical Journal*, 36(6):1389–1401, November 1957.

[13] S. Ramanathan. An algorithm for multicast tree generation in networks with asymmetric links. In *Proceedings of the Conference on Computer Communications (IEEE Infocom)*, pages 337–344, San Fransisco, California, March 1996.

[14] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. RTP: a transport protocol for real-time applications. RFC-1889, January 1996.