STREAMING CACHE PLACEMENT PROBLEMS: COMPLEXITY AND ALGORITHMS

CARLOS A.S. OLIVEIRA, PANOS M. PARDALOS, OLEG A. PROKOPYEV, AND MAURICIO G.C. RESENDE

Abstract. Virtual private networks are often used to distribute live content, such as video or audio streams, to a potentially large number of destinations. Streaming caches (also called splitters) are deployed in these multicast systems to allow content distribution without overloading the network. In this paper, we consider two related combinatorial optimization problems that arise in multicast networks. In the *tree cache placement problem*, the objective is to find a routing tree in which the number of cache nodes needed for (feasible) multicasting is minimized. In a generalization of this problem, called the *flow cache placement problem*, we seek any feasible flow from the source to the destinations that minimizes the number of cache nodes. We prove that these problems are NP-hard using a transformation from Satisfiability. This transformation allows us to give a proof of hardness of approximation by showing that it is gap-preserving. We also consider approximation algorithms, as well as special cases where these problems can be solved in polynomial time.

1. Introduction

Virtual private networks are often used to distribute live content, such as video or audio streams, to a potentially large number of destinations. The process of reserving bandwidth for data sent simultaneously between the involved nodes is resource consuming, and can easily overload the network. One way to decrease congestion is to deploy streaming caches or splitters throughout the system, which is then called a *multicast network*. Each cache receives a single stream and sends out multiple copies of the stream to other caches or destinations. Clearly, deploying caches at every non-source node will minimize the amount of bandwidth that must

Date: December 8, 2004.

Key words and phrases. Multicast networks, streaming service placement, virtual private networks, computational complexity, algorithms.

AT&T Labs Research Technical Report TD-5N2KAU.

be set aside for content distribution. However, since there is a cost associated with the deployment of each cache, there is a tradeoff between the decrease of bandwidth required and the increase in number of caches deployed.

We will assume in this paper that the input network has enough capacity to individually route the stream from the source node to each destination, but may not have enough capacity to route two or more streams simultaneously. Given the capacitated network and the bandwidth requirement of the stream, we wish to locate the minimum number of nodes on which to deploy streaming caches such that each destination can receive a copy of the data and network link capacities are not violated.

In this paper, we consider two combinatorial optimization problems that arise in multicast networks. In the *tree cache placement problem*, the objective is to find a routing tree in which the number of cache nodes needed for multicasting is minimized. In the *flow cache placement problem* (a generalization of the first problem) we seek any feasible flow from the source to the destinations that minimizes the number of cache nodes.

In the remainder of Section 1, we discuss previous work related to cache placement problems. In Section 2, we formally describe the two problems that will be discussed in this paper. In Section 3, we prove that these problems are NP-hard using a transformation from SATISFIABILITY. This transformation allows us to derive a hardness of approximation result in Section 4, by showing that the transformation is gap-preserving. In the same section we discuss improvents to the hardness results, giving a lower bound of $\log |D|$ to the approximation of cache placement problems. In Section 5, we present some approximation algorithms for the cache placement problems. Finally, concluding remarks are made in Section 6.

1.1. **Literature Review.** Applications of multicast routing occur in diverse areas. They range from the deployment of corporate services, such as automatic software updates [10] and groupware [3], to end-user programs for video-conferencing [7] and even game communities [21]. Such problems have also a strong combinatorial appeal, due to the tradeoffs that must be exercised during the design of the network. Examples of problems occurring in multicast networks are the minimum cost multicast tree problem [11, 12, 15, 16], center based tree computation [2, 4, 23], and

the multicast packing problem [22, 27, 28]. A survey by Oliveira and Pardalos [19] gives several other examples of problems and algorithms for multicast networks.

Multicast tree construction is a problem heavily studied by network engineers [11, 12, 15, 16, 27, 28]. The objective is to find a distribution tree, such that source and destination nodes are connected at minimum cost. The problem is a generalization of the well-known Steiner tree problem in graphs [6]. Here, in contrast to our cache location problems, it is assumed that all nodes in the network act as caches. Solution methods for the minimum cost multicast tree include approximation algorithms [5, 26] (mostly based on the corresponding algorithms for Steiner tree) and distributed implementations of heuristics [11, 12, 15].

Shi and Turner [25] discuss multicast networks with a a reduced number of cache servers, organized in a way similar to the one described in the present paper. The authors proposed algorithms for improved routing in this situation, using simulation methods for performance evaluation. In a second paper [24], the same authors studied the minimization of servers required in a multicast network. However they did not consider capacity constraints (as we do in the present paper), since they were more interested in reducing the delay associated with transmission. For this reason, the problem becomes easily reducible to set cover, which has a large number of available algorithms.

The tree cache placement problem, which is one of the main problems discussed in this paper, was introduced by Mao et al. [17]. In that paper, the general problem is proved to be NP-hard using a transformation from Exact Cover by 3-Sets [9]. This is however the only complexity result derived by the authors, and they proceed directly to propose heuristics for the problem. The algorithms were tested empirically, in order to find cache placements under different situations.

Other interesting related work includes the problem of placing replicas of objects in content distribution networks [13, 14]. The objective of this problem, as studied by Kangasharju *et al.* [13], is to minimize the average number of hops traversed by the content being downloaded. The problem has been proved NP-hard by a reduction from bin packing, and some heuristics have been provided for its solution. However, the problem does not consider any form of multicast routing for data delivery.

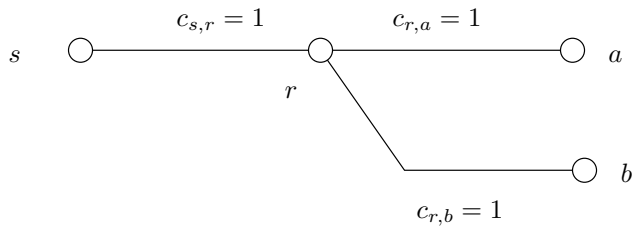

FIGURE 1. Simple example of the tree cache placement problem.

2. Problem Definition

2.1. **The Tree Cache Placement Problem.** Consider a weighted, capacitated network $G = (V, E)$, where V is the set of nodes and E is the set of arcs, with a source node s. Let $T = (V, E_T)$ be a routing tree, where $E_T \subseteq E$, rooted at s and spanning all nodes in V. Node s is required to send a data stream to each node in set $D \subseteq V \setminus \{s\}$. The stream follows the path defined by T from s to the demand nodes and takes up to B units of bandwidth on every edge it traverses. For each demand node $d \in D$, a separate copy of the stream is sent.

Since the network may not have enough capacity to handle all the demand, we deploy stream splitters (also known as caches), at specific nodes in the network. A single copy of the stream is sent from s to a cache node r and from there multiple copies are sent down the tree. The optimization problem is to find a routing tree T and locate a minimum number of cache nodes such that all the data streams can be sent without violating arc capacities.

Figure 1 shows a simple example for this problem, where each edge has unit capacity. In this example, if nodes a and b each require a stream (with $B = 1$) from s, and r is not a cache node, then two units are sent from s to r, one unit is sent from r to a, and one unit is sent from r to b. This leads to an infeasibility on edge (s, r), since it is being traversed by two units, and it has capacity $c_{sr} = 1$. However, if r becomes a cache node, then we can send one unit of data from s to r, one unit from r to a, and one unit from r to b, resulting in a feasible flow. To simplify the formulation of the problem, we can consider without loss of generality, that unit bandwidth is used by each stream. Thus, $B = 1$ in the remainder of this paper.

The *tree cache placement problem* (from now on referred to as TCPP) is defined as follows. Given a graph $G = (V, E)$ with capacities c_{uv} on the edges, a source node $s \in V$, and a subset $D \subseteq V \setminus \{s\}$ representing the destination nodes, find a spanning tree T (which determines the paths followed by a data stream from s to $v \in D$) such that the subset $R \subseteq V \setminus \{s\}$, which represents the cache nodes, has minimum cardinality. For each node $v \in D \cup R$, there must be a data stream coming from some node $w \in R \cup \{s\}$ to v such that the total bandwidth taken by all streams using edge $(i, j) \in T$ does not exceed the edge capacity c_{ij}.

Note that we need to consider only instances where the capacity of each edge is equal to one. To demonstrate this, suppose that we are given an instance with capacities that are integer and greater than one (the network sends only an integer number of streams, thus any fractional capacity may be rounded down to the nearest integer value). Then, create an instance of the problem where each edge (u, v) with capacity k is replicated k times, having as extreme nodes u_i, and v_i, for $i \in \{1, \ldots, k\}$. These new nodes $u_1, \ldots, u_k$ will be linked to the original node u, and similarly to the nodes corresponding to v.

The transformation above increases the size of the problem at most by a factor equal to the largest capacity in the graph G, and can be performed in pseudo-polynomial time (but, in practice, capacities have small values). It is easy to see that a similar transformation can be used for the directed and undirected cases of the problem. Therefore, we will consider in this paper that all instances have unit capacity.

2.2. **The Flow Cache Placement Problem.** An interesting extension of the TCPP arises if we relax the constraint that data streams must be routed from the source node s to the destinations using a tree rooted at s. To see why this extension is interesting, consider the example shown in Figure 2. In this example all edges have capacity equal to one. In a solution to the TCPP on this graph, a stream can be sent through only one of the two edges (s, a) or (s, b) (to avoid cycles). Suppose that we use edges (s, a) and (a, c). Then, to satisfy demand nodes d_1 and d_2, c must be a cache node. However, by relaxing the restriction of routing the streams on a tree, the number of caches can be reduced. For example, if another stream is routed over edges (s, b) and (b, c), no cache node is needed.

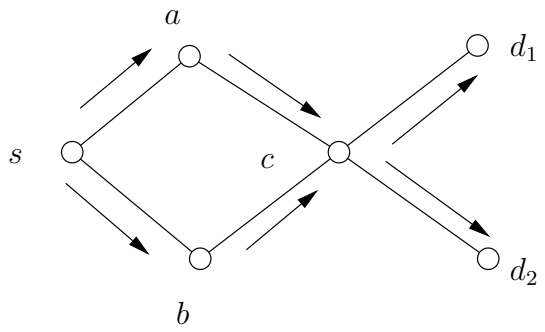

FIGURE 2. Simple example for the flow cache placement problem.

In the *flow cache placement problem* (referred to as FCPP), we seek a feasible flow of data from source s to the set of destinations D, such that the size of the set of cache nodes $R \subseteq V \setminus \{s\}$ is minimized.

3. Complexity of the Cache Placement Problems

In this section, we prove that the TCPP and the FCPP are NP-hard, using a reduction from SATISFIABILITY [9]. Note that the proof is different from the one in [17], which as described in the previous section was based on the EXACT COVER BY 3-SETS. We use a different technique, since we are interested in showing hardness not only for TCPP but also for FCPP, in the directed, and undirected cases. Another advantage of this transformation is that it can be used to derive approximation bounds for the considered problems, as shown in Section 4.

3.1. Complexity of the TCPP.

We first prove that the TCPP is NP-hard. To do this, we use a reduction from the SATISFIABILITY problem (SAT).

Definition 1. *In the Satisfiability Problem, we are given a set of clauses C_1, ..., C_m, where each clause is the disjunction of $|C_i|$ literals (each literal is a variable $x_j \in \{x_1, \ldots, x_n\}$ or its negation $\overline{x}_j$). The objective of SAT is to find if there is a truth assignment for variables $x_1, \ldots, x_n$ such that all clauses are satisfied.*

The decision version of the TCPP (TCPP-D) will be used for a formal reduction of SAT. The TCPP-D is the problem where, given an instance of the TCPP and an integer k, the objective is to determine if there is a feasible solution with size

at most k. In the following theorem we describe a transformation from SAT to the TCPP-D problem, which determines its complexity.

Theorem 1. *The TCPP-D problem is NP-complete.*

Proof. This problem is clearly in NP, since for each instance I it is enough to give the spanning tree and the nodes in R to determine, in polynomial time, if this is a "yes" instance.

We reduce SAT to TCPP-D. Given an instance I of SAT, composed of m clauses $C_1, \ldots, C_m$ and n variables $x_1, \ldots, x_n$, we build a graph $G = (V, E)$, with $c_e = 1$ for all $e \in E$, and choose $k = n$. The set V is defined as

$$V = \{s\} \cup \{x_1, \ldots, x_n\} \cup \{\overline{x}_1, \ldots, \overline{x}_n\} \cup \{T'_1, \ldots, T'_n\} \cup \{T''_1, \ldots, T''_n\} \cup \{T'''_1, \ldots, T'''_n\} \cup \{C_1, \ldots, C_m\},$$

and the set E is defined as

$$E = \bigcup_{i=1}^{n} \{(s, x_i), (s, \overline{x}_i)\} \cup \bigcup_{i=1}^{n} \{(x_i, T'_i), (\overline{x}_i, T'_i)\} \cup \bigcup_{i=1}^{n} \{(x_i, T''_i)\} \cup \bigcup_{i=1}^{n} \{(\overline{x}_i, T'''_i)\} \cup \bigcup_{i=1}^{m} \left\{ \bigcup_{x_j \in C_i} (x_j, C_i) \bigcup_{\overline{x}_j \in C_i} (\overline{x}_j, C_i) \right\}. \tag{1}$$

Figure 1 shows the construction of G for a small SAT instance. Define $D = \{C_1, \ldots, C_m\} \cup \{T'_1, \ldots, T'_n\} \cup \{T''_1, \ldots, T''_n\} \cup \{T'''_1, \ldots, T'''_n\}$. The objective of destination nodes T'_i, T''_i, and T'''_i is to saturate the arcs leaving s and force one of the literals x_i or $\overline{x}_i$ to be chosen as a cache node. Each node C_i forces the existence of at least one cache among the nodes corresponding to literals appearing in clause C_i.

Suppose that the solution of the resulting TCPP-D problem is true. Then, we assign variable x_i to true if node x_i is in R, otherwise we set x_i to false. This assignment is well-defined, since exactly one of the nodes $x_i, \overline{x}_i$ must be selected. Clearly, this truth assignment satisfies all clauses C_i, because the demand of each node C_i is satisfied by at least one node corresponding to literals appearing in clause C_i.

Conversely, if there is a truth assignment Γ that makes the SAT formula satisfiable, we can use it to define the nodes which will be caches, and, by construction of

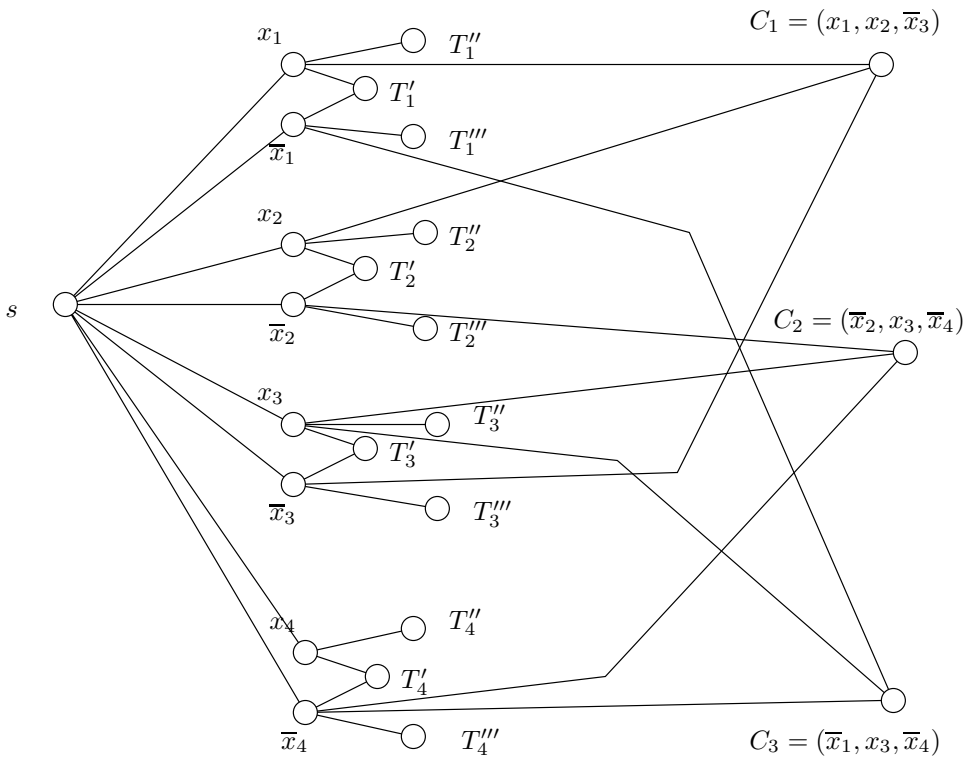

FIGURE 3. Small graph G created in the reduction given by Theorem 1. In this example, the SAT formula is $(x_1 \vee x_2 \vee \overline{x}_3) \wedge (\overline{x}_2 \vee x_3 \vee \overline{x}_4) \wedge (\overline{x}_1 \vee x_3 \vee \overline{x}_4)$.

G, all demands will be satisfied. Finally, the resulting construction is polynomial in size, thus SAT reduces in polynomial time to TCPP-D. □

Corollary 1. *The TCPP is NP-hard.*

3.2. **Complexity of the FCPP.** We can use the transformation from SAT to TCPP to show that FCPP is also NP-hard. In the case of directed edges, this is simple, since given a graph G provided by the reduction, we can give an orientation of G from source to destinations. This is stated in the next theorem.

Theorem 2. *The FCPP is NP-hard if the instance graph is directed.*

Proof. The proof is similar to the proof of Theorem 1. We need just to make sure that the polynomial transformation given for the TCPP-D also works for a decision

version of the FCPP. Given an instance of SAT, let G be the corresponding graph found by the reduction. We orient the edges of G from s to the destination nodes in D, i.e. use the implicit orientation given in (1). It can be checked that in the resulting instance the number of cache nodes cannot be reduced by sending additional flow on edges other than the ones forming the tree in the solution of TCPP. Thus, the resulting R is the same, and FCPP is NP-hard in this case. □

Next we prove a slightly modified theorem for the undirected version. To do this we need the following variant of SAT.

Definition 2. 3SAT(5): *Given an instance of* SATISFIABILITY *with at most three literals per clause and such that each variable appears in at most five clauses, is there a truth assignment that makes all clauses true?*

The 3SAT(5) is well known to be NP-complete [9].

Theorem 3. *The FCPP is NP-hard if the instance graph is undirected.*

Proof. When the instance of FCPP is undirected, it may be possible that some of the destinations T_i', T_i'', or T_i''' are being satisfied by flow coming from nodes C_j connected to their respective x_i, $\overline{x}_i$ nodes. What can be done to prevent this is bounding the number of occurrences of each variable and add enough absorbing destinations to the subgraph corresponding to that variable. We do this by reduction from 3SAT(5). The reduction is essentially the same as the reduction from SAT to TCPP, but now for each variable x_i we have nodes x_i, $\overline{x}_i$, T_i', T_i'', and T_i^k, for $1 \leq k \leq 6$ (see Figure 4). Also, for each variable x_i we have edges (s, x_i), $(s, \overline{x}_i)$, (x_i, T_i'), $(\overline{x}_i, T_i'')$, (x_i, T_i^k), $(\overline{x}_i, T_i^k)$, for $1 \leq k \leq 6$.

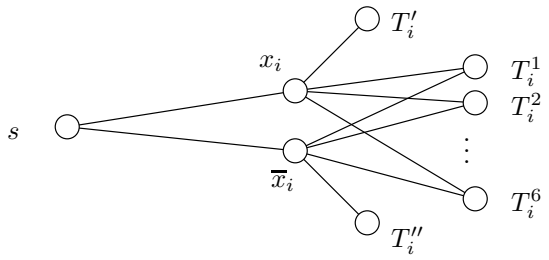

FIGURE 4. Part of the transformation used by the FCPP.

We claim that in this case for each pair of nodes x_i, $\overline{x}_i$, one of them must be a cache node (which says that the corresponding variable in 3SAT(5) is true or false). This occurs because from the eight destinations not corresponding to clauses (T_i', T_i'', and T_i^k, for $1 \leq k \leq 6$) attached to x_i, $\overline{x}_i$, two can be directly satisfied from s without caches. However, the remaining six cannot be satisfied from nodes C_j linked to the current variable nodes, because there are at most five such nodes. Thus, we must have one cache node at x_i or $\overline{x}_i$, for each variable x_i. It is clear that these are the only cache nodes needed to make all destinations satisfied. This gives us the correct truth assignment for the original 3SAT(5) instance. Conversely, any non-satisfiable formula will transform to a FCPP instance which needs more than n cache nodes to satisfy all destinations. Thus, the decision version of FCPP is NP-complete, and this implies the theorem. □

3.3. **Polynomially Solvable Cases.** Although the TCPP and the FCPP are NP-hard in general, some particular classes of instances can be solved in polynomial time. For instance, if G is the complete graph K^n, then the optimal solution is simply a star graph with s at the center, and $R = \emptyset$.

A less trivial example of graphs where the problem is solvable in polynomial time is given by a tree. If the graph is a tree with n nodes, then the set of required caches is implied by the edges appearing on the tree, and the optimal solution is completely determined. Algorithm 1 shows how to find this optimal set for a given tree T. The algorithm works recursively. Initially it finds the demand for all leaves of T. Then it goes up the tree determining at each step if the current node needs to become a cache node. The correctness of this method is proved below.

Theorem 4. *Given an instance of the TCPP, in which the input network is a tree T, an optimal solution is given by Algorithm 1.*

Proof. The proof is by induction on the height h of a tree analyzed when Algorithm 1 reaches line (1). If $h = 1$ then the number of cache nodes is clearly equal to zero. Assume that the theorem is true for trees with height $h > 1$. If the capacity of the arc (p, v) is greater than the demand at v, then there is no need of a new cache node, and therefore the solution remains optimal. If, on the other hand, (p, v) does not have enough capacity to satisfy all demand at v, then we do not have a choice

Algorithm 1: Find the optimal R for a fixed tree.

```
Input: A tree T.
Output: A set R of cache nodes.
forall v ∈ V do
    if v ∈ D then demand(v) ← 1
    else demand(v) ← 0
call findR(s)
return R

procedure findR(v)
begin
    forall w such that (v, w) ∈ T do call findR(w)
1   if v = s then return R
    else p ← parent(v)
    if c_{p,v} < demand(v) then
        R ← R ∪ {v}
        demand(p) ← demand(p) + 1
    else demand(p) ← demand(p) + demand(v)
end
```

other than making v a cache node. Combining this with the assumption that the solution for all children of v is optimal, we conclude that the new solution for a tree of height $h+1$ is also optimal. □

Turning now to the FCPP, a case that is solvable in polynomial time occurs when $k = 0$, i.e., the problem of determining if any cache node is needed at all. The solution for this problem is given by the following algorithm. Initially, run the maximum flow algorithm from node s to all nodes in D (this can be accomplished, for example, by creating a dummy destination node d and linking all nodes $v \in D$ to d by arcs with capacity equal to 1). If the maximum flow from s reaches each node in D, then the answer is true, since no cache node is needed to satisfy the destinations. Otherwise, the answer must be false because then at least one cache node is needed to satisfy all nodes in D.

4. Hardness of Approximation Results

The transformation used in Theorem 1 provides a method for proving a hardness of approximation result for the TCPP and FCPP. We employ standard techniques, based on the gap-preserving transformations. To do this, we use an optimization version of 3Sat(5).

Definition 3. Max-3Sat(5): *Given an instance of* 3Sat(5)*, find the maximum number of clauses that can be satisfied by any truth assignment.*

Definition 4. *For any* ϵ*,* $0 < \epsilon < 1$*, an approximation algorithm with guarantee* ϵ *(or equivalently, an* ϵ*-approximation algorithm) for a maximization problem* Π *is an algorithm* A *such that, for any instance* $I \in \Pi$*, the resulting cost* $A(I)$ *of* A *applied to instance* I *satisfies* $\epsilon \cdot OPT(I) \leq A(I)$*, where we denote by* $OPT(I)$ *the cost of the optimum solution. For minimization problems,* $A(I)$ *must satisfy* $A(I) \leq \epsilon \cdot OPT(I)$*, for any fixed* $\epsilon > 1$*.*

The following theorem from [1] is very useful to prove hardness of approximation results.

Theorem 5. *There is a polynomial time reduction from SAT to* Max-3Sat(5) *which transforms formula* ϕ *into a formula* ϕ' *such that, for some fixed* ϵ *(*ϵ *is in fact determined in the proof of the theorem),*

- *if* ϕ *is satisfiable, then* $OPT(\phi') = m$*, and*
- *if* ϕ *is not satisfiable, then* $OPT(\phi') < (1 - \epsilon)m$*,*

where m *is the number of clauses in* ϕ'*.*

In the following theorem we use this fact to show the hardness of approximating the TCPP.

Theorem 6. *The transformation used in the proof of Theorem 1 is a gap-preserving transformation from* Max-3SAT(5) *to TCPP. In other words, given an instance* ϕ *of* Max-3SAT(5) *with* m *clauses and* n *variables, we can find an instance* I *of TCPP such that*

- *If* $OPT(\phi) = m$ *then* $OPT(I) = n$*; and*
- *If* $OPT(\phi) \leq (1 - \epsilon)m$ *then* $OPT(I) \geq (1 + \epsilon_1)n$*,*

where ϵ is given in Theorem 5 and $\epsilon_1 = \epsilon/15$.

Proof. Suppose that ϕ is an instance of MAX-3SAT(5). Then, we can use the transformation given in the proof of Theorem 1 to construct a corresponding instance I of TCPP. If ϕ has a solution with $OPT(\phi) = m$, where m is the number of clauses, then by Theorem 1, we can find a solution for I such that $OPT(I) = n$.

Now, if $OPT(\phi) \leq (1-\epsilon)m$, then there are at least ϵm clauses unsatisfied. In the corresponding instance I, we have at least n cache nodes due to the constraints from nodes T_i', T_i'', and T_i''', for $1 \leq i \leq n$. These cache nodes satisfy at most $(1-\epsilon)m$ destinations corresponding to clauses. Let U be the set of unsatisfied destinations. The nodes in U can be satisfied by setting one extra cache (in a total of two, for nodes x_j and $\overline{x}_j$) for at least one variable x_j appearing in the clause corresponding to c_i, for all $c_i \in U$.

Thus, the number of extra cache nodes needed to satisfy U is at least $|U|/5$, since a variable can appear in at most 5 clauses. We have

$$OPT(I) \geq n + |U|/5 \geq n + \epsilon m/5 \geq (1 + \epsilon/15)n.$$

The last inequality follows from the trivial bound $m \geq n/3$. The theorem follows by setting $\epsilon_1 = \epsilon/15$. □

Definition 5. *A PTAS (Polynomial Time Approximation Scheme) for a minimization problem Π is an algorithm that, for each $\epsilon > 0$ and instance $I \in \Pi$, returns a solution $A(I)$, such that $A(I) \leq (1+\epsilon)OPT(I)$, and A has running time polynomial in the size of I, depending on ϵ (see, e.g.,* [20, page 425]*).*

Corollary 2. *Unless $P = NP$, the TCPP cannot be approximated by $(1+\epsilon_2)$ for any $\epsilon_2 \leq \epsilon_1$, where ϵ_1 is given in Theorem 6, and therefore there is no polynomial time approximation scheme (PTAS) for the TCPP.*

Proof. Given an instance ϕ of SAT, we can use the transformation given in Theorem 5, coupled with the transformation given in the proof of Theorem 1, to give a new polynomial transformation τ from SAT to TCPP. Now, let I be the instance created by τ on input ϕ. Suppose there is an ϵ_2 approximation algorithm A for TCPP, with $0 \leq \epsilon_2 \leq \epsilon_1$. Then, when A runs on an instance I constructed by τ from a satisfiable formula ϕ, the result must have cost $A(I) \leq (1+\epsilon_2)n < (1+\epsilon_1)n$.

Otherwise, if ϕ is not satisfiable, then the result given by this algorithm must be greater than $(1 + \epsilon_1)n$, because of the gap introduced by τ. Thus, if there is an ϵ_2-approximation algorithm, then we can decide in polynomial time if a formula ϕ is satisfiable or not. Assuming P $\neq$ NP, there is no such algorithm.

The fact that there is no PTAS for the TCPP is a consequence of this result and the definition of PTAS. □

The above theorem and corollary can be easily extended to the FCPP. The fact that the same transformation can be used for both problems can be used to demonstrate the hardness of approximation result for the FCPP as well. We state this as a corollary.

Corollary 3. *Unless $P = NP$, the FCPP has no PTAS.*

Proof. The transformation from SAT to FCPP is identical, so Theorem 6 is also valid for the FCPP. This implies that the FCPP has no PTAS, unless P = NP. □

4.1. **An Improved Lower Bound.** In this section we improve the hardness of approximation result for the cache placement problem, showing that there is no approximation algorithm for the problems with performance guarantee better than $\log k$, where k is the number of destinations. The proof appeared first in [18], but we extend it here for the undirected version of the problems. The technique used is based on a reduction from the Set Cover problem.

Set Cover: Given a ground set $T = t_1, \ldots, t_n$, with subsets $S_1, \ldots, S_m \subset T$, find the minimum cardinality set $C \subseteq \{1, \ldots, m\}$ such that $\bigcup_{i \in C} S_i = T$.

It is known [8] that Set Cover does not have approximation algorithms for any guarantee better than $O(\log n)$. Thus, if we find a transformation from Set Cover to SCPP that preserves approximation, we can provide a similar result for SCPP. We show how this transformation, which will be represented by $\phi : \text{SC} \rightarrow \text{SCPP}$, can be done.

For each instance I_{SC} of set cover, we must find a corresponding instance I_{SCPP} of the SCPP. The instance I_{SC} is composed of sets T and $S_1, \ldots, S_m$ as shown above. The transformation consists of defining a capacitated graph G with a source

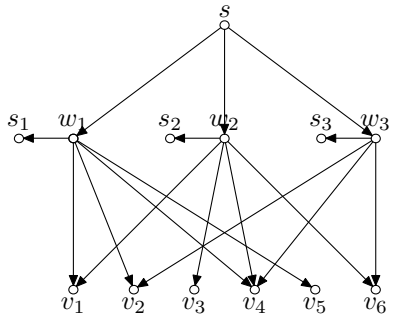

FIGURE 5. Example for the transformation in Theorem 7.

and a set D of destinations. Let G be the graph composed of the following nodes:

$$V = \{s\} \cup \{w_1, \ldots, w_m\} \cup \{v_1, \ldots, v_n\} \cup \{s_1, \ldots, s_m\}.$$

Also, let the edges E of the graph G be

$$E = \{(w_j, v_i) \mid t_i \in S_j\} \cup \bigcup_{i=1}^{m} \{(s, w_i)\} \cup \bigcup_{i=1}^{m} \{(w_i, s_i)\}.$$

In the instance of SCPP, the set of destination nodes D is given by

$$D = \{v_1, \ldots, v_n\} \cup \{s_1, \ldots, s_m\},$$

and s is the source node. Thus, there is a one to one correspondence between nodes w_i and sets S_i, for $1 \leq i \leq m$. There is also a one to one correspondence between nodes v_i and ground elements $t_i \in T$, for $1 \leq i \leq n$. There is a directed edge between the source and each node w_i, and between nodes w_i and nodes representing elements appearing in the set S_i. Nodes w_i are also linked to each s_i. Finally, each edge e has capacity $c_e = 1$. See an example of such reduction in Figure 5. The ground set in this example is $T = \{t_1, \ldots, t_6\}$, and the subsets are $S_1 = \{t_1, t_2, t_4, t_5\}$, $S_2 = \{t_1, t_3, t_4, t_6\}$, and $S_3 = \{t_2, t_4, t_6\}$.

Theorem 7. *The transformation described above is a polynomial time reduction from* SET COVER *to SCPP.*

Proof. Let I_{SC} be the instance of SET COVER and I_{SCPP} the corresponding instance of the SCPP. It is clear that the transformation is polynomial, since the number of edges and nodes is given by a constant multiple of the number of elements and sets in the instance of SET COVER. We must prove that I_{IS} and I_{SCCP}

have equivalent optimal solutions. Let S' be an optimal solution for I_{SC}. First we note that the destination nodes s_i, $1 \leq i \leq m$, can be reached only from nodes w_i, and therefore each s_i must be satisfied with flow coming from w_i. Thus, each node s_i saturates the corresponding w_i, which means that to satisfy any other node from w_i we must make it a cache node. Then, we can clearly make $R = \{w_i \mid i \in S'\}$, and serve all remaining destinations in $v_1, \ldots, v_n$, by definition of S'. Each node in R will be a cache node, and therefore R is a solution for I_{SCPP}. This solution must be optimal, because otherwise we could use a smaller solution R' to construct a corresponding set $S'' \subset \{1, \ldots, m\}$ with $|S''| < |S'|$, covering all elements of T, and therefore contradicting the fact that S' is an optimum solution for the SC instance. Thus, the two instances I_{SC} and I_{SCPP} have equivalent optimal solutions. □

Corollary 4. *Given an instance I of SC, and the transformation ϕ described above, then we have $OPT(I) = OPT(\phi(I))$.*

The following theorem, proved by Feige [8] is the base for our main result.

Theorem 8 ([8])**.** *If there is some $\epsilon > 0$ such that a polynomial time algorithm can approximate set cover within $(1 - \epsilon) \log n$, then NP$\subset TIME(n^{O(\log \log n)})$.*

This theorem implies that finding approximate solutions with guarantee better than $(1 - \epsilon) \log n$ for Set Cover is equivalent to solving any problem in NP in sub-exponential time. It is strongly believed that this is not the case. We use this theorem and the reduction above to give a related bound for the approximation of SCPP. To do this, we need a gap *preserving transformation* from SC to SCPP, as stated in the following lemma.

Lemma 1. *If I is an instance* Set Cover*, then the transformation ϕ from SC to SCPP described above is* gap preserving*, that is, it has the following property:*

(a) *If $OPT(I) = k$ then $OPT(\phi(I)) = k$; and*
(b) *If $OPT(I) \geq k \cdot \log n$ then $OPT(\phi(I)) \geq k(\log |D| - 1)$,*

where k is a fixed value, depending on the instance.

Proof. Part (a) is a simple consequence of Corollary 4. Now, for part (b), note that according to [8] the hardness of approximation result is valid for instances

with number of subsets less or equal to n. Consequently, in the instance of SCPP created by transformation ϕ, $|D| = m + n \leq 2n$. Thus, we have

$$\log |D| \leq \log(2n) = \log n + 1.$$

This implies that

$$OPT(\phi(I)) \geq k \log n \geq k(\log |D| - 1). \qquad \square$$

The reduction shown in Theorem 7 is gap preserving, since it maintains an approximation gap, introduced by the instances of Set Cover. We see that the approximation guarantee $\log |D|$ implied by the previous result is asymptotically optimal. This is formalized in the next theorem.

Theorem 9. *If there is some $\epsilon > 0$ such that a polynomial time algorithm A can approximate SCPP within $(1-\epsilon) \log k$, where $k = |D|$, then $NP \subset TIME(n^{O(\log \log n)})$.*

Proof. Suppose that an instance I of the SC is given. The transformation ϕ described above can be used to find an instance $\phi(I)$ of the SCPP. Then, A can be used to solve the problem for instance $\phi(I)$. According to Lemma 1, transformation ϕ reduces any gap of $\log n$ to $\log k$. Thus, with such an algorithm one can differentiate between instances I with a gap of $\log n$. But this is not possible in polynomial time, according to [8, Theorem 10] unless NP$\subset TIME(n^{O(\log \log n)})$. $\square$

4.2. **Extending the Approximation Bound.** It is interesting to observe that the transformation presented above is not enough to show the hardness result for the undirected version of the SCPP problem. In this case, the difference is that an algorithm can send flow to one of the ground set destinations, turning it into a cache, and from there satisfying other destinations corresponding to subsets. Such interactions could allow some destinations to be served without increasing the number of cache nodes.

To solve this problem, we employ a technique similar to the one used in the transformation from SAT to FCPP in Section 3.2: use additional destinations that need to be satisfied at each node w_i, in order to "saturate" edges coming from a node v_i. Thus, we can substitute the edge (s_i, w_i) by a set of edges $(s_{i0}, w_i), \ldots, (s_{in}, w_i)$, for $i \in \{1, \ldots, m\}$, where the s_{ij}'s are destination nodes. Doing this, we know that the $n+1$ nodes $s_{i0}, \ldots, s_{in}$ cannot all be satisfied by streams coming from the nodes

$v_1, \ldots, v_n$, and therefore they perform the same task that the edge (s_i, w_i) does in Lemma 1.

The extension of this result to the directed and undirected versions of FCPP is trivial: the same constructions used above work for the FCPP as well, since it is just a generalization of the SCPP.

5. Approximation Algorithms

In this section, we present approximation algorithms for the TCPP and FCPP and analyze their approximation guarantee. To simplify our results, we use the notation $A(I) = |R \cup \{s\}|$, where R is the set of cache nodes found by algorithm A applied to instance I. Also, $OPT(I) = |R^* \cup \{s\}|$, where R^* is an optimal set of cache nodes for instance I. Note that $A(I) \geq 1$ and $OPT(I) \geq 1$, which makes Definition 4 valid for our problems.

5.1. A Simple Algorithm for TCPP.

It is easy to construct a simple approximation algorithm for the TCPP. We denote by $\delta_G(v)$ the degree of node v in the network G.

Algorithm 2: Spanning Tree Algorithm

Input: Network G, destinations D, source s.
Output: A set R of cache nodes.
Step 1: Construct a spanning tree T of G
Step 2: Remove recursively all leaves of T which are not in $D \cup \{s\}$
Step 3: Let S_1 be the set of internal nodes v with $\delta_T(v) > 2$
Step 4: Let S_2 be the set of internal nodes v with $\delta_T(v) = 2$ and $v \in D$
Step 5: Return $R = S_1 \cup S_2$.

The correctness of the algorithm is shown in the next lemma.

Lemma 2. *Algorithm 2 returns a feasible solution to the TCPP.*

Proof. The operation in Step 2 maintains feasibility, since leaves cannot be used to reach destinations. The result R includes all internal nodes v with $\delta_T(v) > 2$, and all internal nodes v with $\delta_T(v) = 2$ and $v \in D$. It suffices to prove that if $\delta_T(v) = 2$ and $v \notin D$ then v is not needed in R.

Suppose that v is an internal node with $\delta_T(v) = 2$ and $v \notin D$. If the number of destinations down the tree from v is equal to 1, then v does not need to be a cache. Now, assume that the number of cache nodes down the tree from v is two or more. Then, there are two cases. In the first case, there is a node w, between v and the destinations, with $\delta_T(w) > 2$. In this case, w is in R, and we need just to send one unit of flow from v to w, thus v does not need to be in R. In the second case, there must be some destination w with $\delta_T(w) = 2$ between v and the other destinations. Again, in this case w will be included in R from S_2. Thus v does not need to be in R. This shows that R is a feasible solution to the TCPP. □

Lemma 3. *Algorithm 2 gives an approximation guarantee of $|D|$.*

Proof. Let us partition the set of destinations among D_1 and D_2, where $D_1 = D \setminus S_2$ and $D_2 = S_2$. Denote by D' the set of destinations which are leaves in T. Initially, note that for any tree the number of nodes v with degree $\delta(v) > 2$ is at most $|L| - 2$, where L is the set of nodes with $\delta(v) = 1$ (the leaves). Since L, in this case, is $D' \cup \{s\} \subseteq D_1 \cup \{s\}$, then $|S_1| \leq |D_1 \cup \{s\}| - 2$. Thus,

$$|R| = |S_1 \cup S_2| \leq |D_1 \cup \{s\}| - 2 + |D_2| = |D| - 1,$$

and

$$A(I) = |R| + 1 \leq |D| \leq |D| OPT(I),$$

since $OPT(I) \geq 1$. □

Let $\Delta = \Delta(G)$ be the maximum degree of G. In the case in which all capacities c_e, for $e \in E(G)$, are equal to one, we can give a better analysis of the previous algorithm with an improved performance.

Theorem 10. *Algorithm 1 is a $\min\{\Delta(G), |D|\}$-approximation algorithm.*

Proof. The key idea to note is that if $c_e = 1$ for all $e \in E$, then $\lceil |D|/\Delta \rceil \leq OPT(I)$, for any instance I of the TCPP. This happens because each cache node (as well as the source) can serve at most Δ destinations. Let $A(I)$ be the value returned by Algorithm 2 on instance I. We know from the previous analysis of Lemma 3 that $A(I) \leq |D|$. Thus $A(I) \leq \Delta OPT(I)$. The theorem follows, since we know that this is also an $|D|$-approximation algorithm. □

5.2. **A Flow-based Algorithm for FCPP.** In this section, we present an approximation algorithm for the FCPP. The algorithm is based on the idea of sending flow from the source to destination nodes. We show that this algorithm performs at least as well as the previous algorithm for the TCPP. In addition, we show that for a special class of graphs, this algorithm finds the optimal solution. Therefore, for this class of graphs the FCPP is solvable in polynomial time.

Let $f(x, y) \in \mathbb{R}^+$ be the amount of flow sent on edge (x, y), for $(x, y) \in E$. A flow is *feasible* for the SCPP if it satisfies the flow conservation constraints, i.e., the amount of data entering a node is the same amount leaving minus its demand; unless the node is a cache, in which case any amount of data can be sent. Let $F(f, s, t) = \sum_{v \in V} f(s, v)$ be the total flow sent from node s. We assume that s can send at most $\sum_{(s,v) \in E} c_{sv}$ units of flow, and t can receive at most $\sum_{(u,t) \in E} c_{ut}$ units of flow. We say that a feasible flow f is the *maximum flow* from s to t if there is no feasible flow f' such that $F(f', s, t) > F(f, s, t)$. A node v is *reached* from s by flow f if $\sum_{w \in V} f(w, v) > 0$. It is well known that when f is the maximum flow, then $F(f, s, t) = C(s, t)$, where $C(s, t)$ represents the minimum capacity of any set of edges separating s from t in G (the *minimum cut*). We also use the notation $C(U, \overline{U})$ to denote the total capacity of edges linking nodes in U to nodes in $\overline{U}$, where $U \subset V$ and $\overline{U} = V \setminus U$.

Denote any feasible flow starting from node v by f_v. The algorithm works by finding the maximum flow from s to all nodes in D. If the total cost of this maximum flow is $F(f_s, s, D) \geq |D|$, then the problem is solved, since all destinations can be reached without cache nodes. Otherwise, we put all nodes reachable from s in a set Q. Then we repeat the following steps until $D \setminus Q$ is empty:

- For all nodes $v \in Q$, compute the maximum flow f_v from v to D.
- Find the node v^* such that f_{v^*} is maximum.
- Add v^* to R and add to Q all nodes reachable from v^*.
- Reduce the capacity of the edges in E by the amount of flow used by f_{v^*}.

These steps are described more formally in Algorithm 3.

In the following theorem, we show that the running time of this algorithm is polynomial and depends on the time needed to find the maximum flow. Denote by $C^M(G)$ the maximum value of the minimum cut between any pair of nodes

Algorithm 3: Flow algorithm.

```
Q ← {s}
while D \ Q ≠ ∅ do
    forall v ∈ Q do
        find the maximum flow f_v from v to D \ Q
    Let v* be the node such that F(f_{v*}, s, D) is maximum
    R ← R ∪ {v*}
    Add to Q the nodes reached by f_{v*}
    for each edge (u, v) ∈ E do
        Reduce capacity c_{u,v} by f_{v*}(u, v)
```

$v, w \in V(G)$, i.e.

$$C^M(G) = \max_{v,w \in V} C(v, w).$$

Similarly, we define

$$C^m(G) = \min_{v,w \in V} C(v, w).$$

Theorem 11. *Algorithm 3 has running time equal to $\mathcal{O}(n \cdot |D| \cdot T^{mf}/C^m(G))$, where T^{mf} is the time needed to run the maximum flow algorithm.*

Proof. The most costly operations in this algorithm are calls to the maximum flow algorithm. Therefore we count the number of calls. Note that, at each of the N iterations of the while loop, a new element is added to the set of cache nodes. Thus, $A(I)$ is equal to the number of such iterations.

Let v_i be the node added to the set of cache nodes at iteration i, and Q^i be the content of set Q at iteration i of Algorithm 3. At each step, the number of elements of D found by the algorithm is, according to the maximum-flow/minimum-cut theorem, equal to the minimum cut from v_i to the remaining nodes in $D \setminus Q^i$ (recall that all demands are unitary). Then,

$$|D| = \sum_{i=1}^{N} C(v_i, D \setminus Q^i) \geq \sum_{i=1}^{N} \min_{w \in D} C(v_i, w) \geq A(I) \min_{v,w \in V} C(v, w).$$

Thus, we have

$$N = A(I) \leq \frac{|D|}{C^m(G)}. \tag{2}$$

At each iteration of the while loop, the number of calls to the maximum flow algorithm is at most n. The total number n_c of such calls is given by $n_c \leq n|D|/C^m(G)$. Thus, the running time of Algorithm 3 is $\mathcal{O}(n \cdot |D| \cdot T^{mf}/C^m(G))$. □

Based on the performance analysis just shown, the following theorem gives an approximation guarantee for Algorithm 3.

Theorem 12. *Algorithm 3 is a k-approximation algorithm, where*

$$k = C^M(G)/C^m(G).$$

Proof. If we denote by R the set of cache nodes in the optimal solution, we have

$$|D| \leq \sum_{v_i \in R \cup \{s\}} C(v_i, D) \leq \sum_{i \in R \cup \{s\}} \max_{v,w \in V} C(v, w) = OPT(I)C^M(G). \tag{3}$$

Combining inequalities (2) and (3) results in

$$A(I) \leq \frac{C^M(G)}{C^m(G)} OPT(I).$$

□

Note that the quantity $C^M(G)/C^m(G)$ can become large. However, for some types of networks, the preceding algorithm gives us a better understanding of the problem. For example, if the network has maximum degree $\Delta(G)$, then it is easy to see that

$$\frac{C^M(G)}{C^m(G)} \leq \Delta(G).$$

If the edge capacity is not fixed, then $C^M(G)/C^m(G)$ becomes at most $\Delta(G) \cdot c^M/c^m$, where c^M represents the maximum capacity and c^m the smallest capacity of edges in G.

6. Concluding Remarks

In this paper we presented and analyzed two combinatorial optimization problems that arise in multicast networks, the tree cache placement problem and its flow-based, generalized version, the flow cache placement problem. We prove that both problems are NP-hard and that they are not solvable by any PTAS unless P = NP. We also develop approximation algorithms for both TCPP and FCPP.

An open question for the problems discussed in this paper concerns the development of approximation algorithms with good performance guarantee. We believe that a performance guarantee of $\log |D|$ is possible, based on the reductions from Set Cover previously presented. For problems similar to Set Cover, the greedy algorithm is known to perform well. However, in this case there is no easy way of defining a cover for the destination nodes, and therefore the use of the greedy algorithm is not immediate. It would be interesting to find algorithms with better approximation guarantee, or improved hardness of approximation results.

References

[1] S. Arora and C. Lund. Hardness of approximations. In D. Hochbaum, editor, *Approximation Algorithms for NP-hard Problems*. PWS Publishing, 1996.

[2] A. Ballardie, P. Francis, and J. Crowcroft. Core-based trees (CBT) – an architecture for scalable inter-domain multicast routing. *Computer Communication Review*, 23(4):85–95, 1993.

[3] G. V. Chockler, N. Huleihel, I. Keidar, and D. Dolev. Multimedia multicast transport service for groupware. In *TINA Conference on the Convergence of Telecommunications and Distributed Computing Technologies*, pages 43–54, 1996.

[4] S. Deering, D. Estrin, D. Farinacci, V. Jacobson, C.-G. Liu, and L. Wei. An architecture for wide-area multicast routing. *Computer Communication Review*, 24(4):126–135, 1994.

[5] M. Doar and I. Leslie. How bad is naive multicast routing. In *Proceedings of the IEEE INFOCOM*, pages 82–89, Los Alamitos, Calif, USA, 1993. IEEE Comput Soc Press.

[6] D.-Z. Du, B. Lu, H. Ngo, and P. M. Pardalos. Steiner tree problems. In C. Floudas and P. Pardalos, editors, *Encyclopedia of Optimization*, volume 5, pages 227–290. Kluwer Academic Publishers, 2001.

[7] H. Eriksson. MBONE: The multicast backbone. *Communications of ACM*, 37(8), 1994.

[8] U. Feige. A threshold of ln n for approximating set cover. *Journal of the ACM*, 45(4):634–652, 1998.

[9] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-completeness*. W.H. Freeman and Company, 1979.

[10] L. Han and N. Shahmehri. Secure multicast software delivery. In *IEEE 9th International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WET ICE'00)*, pages 207–212, 2000.

[11] Y. Im, Y. Lee, and Y. Choi. A delay constrained distributed multicast routing algorithm. *Computer Communications*, 20(1), 1997.

[12] X. Jia. A distributed algorithm of delay-bounded multicast routing for multimedia applications in wide area networks. *IEEE/ACM Transactions on Networking*, 6(6):828–837, 1998.

[13] J. Kangasharju, J. W. Roberts, and K. W. Ross. Object replication strategies in content distribution networks. In *6th International Web Content Caching and Distribution Workshop*, 2001.

[14] M. Karlsson and M. Mahalingam. Do we need replica placement algorithms in content delivery networks? In *Proceedings of 7th International Web Content Caching and Distribution Workshop*, 2002.

[15] V. Kompella, J. Pasquale, and G. Polyzos. Multicasting for multimedia applications. In *Proceedings of IEEE INFOCOM'92*, pages 2078–2085, 1992.

[16] V. Kompella, J. Pasquale, and G. Polyzos. Multicast routing for multimedia communication. *IEEE/ACM Trans. Networking*, 1(3):286–292, 1993.

[17] Z. Mao, D. Johnson, O. Spatscheck, J. van der Merwe, and J. Wang. Efficient and robust streaming provisioning in VPNs. In *Proceedings of WWW2003*, pages 118–127. ACM, 2003.

[18] C. A. Oliveira. *Optimization Problems in Telecommunications and the Internet*. PhD thesis, Department of Industrial and Systems Engineering, University of Florida, 2004.

[19] C. A. Oliveira and P. M. Pardalos. A survey of combinatorial optimization problems in multicast routing. To appear in Computers and Operations Research, 2004.

[20] C. H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization*. Prentice Hall, 1982.

[21] J. Park and C. Park. Development of a multi-user & multimedia game engine based on TCP/IP. In *Proceedings of IEEE Pacific Rim Conference on Communications Computers and Signal Processing*, pages 101–104, 1997.

[22] V. Priwan, H. Aida, and T. Saito. The multicast tree based routing for the complete broadcast multipoint-to-multipoint communications. *IEICE Transactions on Communications*, E78-B(5), 1995.

[23] H. F. Salama, D. S. Reeves, and Y. Viniotis. Shared multicast trees and the center selection problem: A survey. Tr-96/27, Dept. of Electrical and Computer Engineering, NCSU, 1996.

[24] S. Shi and J. S. Turner. Placing servers in overlay networks. In *Proc. of International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPETS)*, 2002.

[25] S. Shi and J. S. Turner. Routing in overlay multicast networks. In *Proc. of IEEE INFOCOM*, New York City, 2002.

[26] D. Wall. *Mechanisms for broadcast and selective broadcast*. PhD thesis, Computer Science Department, Stanford University, 1982.

[27] C.-F. Wang, C.-T. Liang, and R.-H. Jan. Heuristic algorithms for packing of multiple-group multicasting. *Computers & Operations Research*, 29(7):905–924, 2002.

[28] B. Yener, S. Chen, and O. Günlük. Optimal packing of group multicastings. In *Proc. IEEE INFOCOM'98*, 1998.

(C.A.S. Oliveira) School of Industrial Engineering and Management, Oklahoma State University, 322 Engineering North, Stillwater, OK 74078, USA.

E-mail address, C.A.S. Oliveira: coliv@okstate.edu

(P.M. Pardalos) Department of Industrial and Systems Engineering, University of Florida, 303 Weil Hall, Gainesville, FL 32611, USA.

E-mail address, P.M. Pardalos: pardalos@ufl.edu

(O.A. Prokopyev) Department of Industrial and Systems Engineering, University of Florida, 303 Weil Hall, Gainesville, FL 32611, USA.

E-mail address, O.A. Prokopyev: prokopyev@ufl.edu

(M.G.C. Resende) Internet and Network Systems Research, AT&T Labs Research, 180 Park Avenue, Room C241, Florham Park, NJ 07932 USA.

E-mail address, M.G.C. Resende: mgcr@research.att.com