P2P Streaming Capacity

Sudipta Sengupta[‡], Shao Liu[*], Minghua Chen[•], Mung Chiang[*], Jin Li[‡], and Philip A. Chou[‡]
[‡]Microsoft Research, Redmond, WA, USA
[*]Dept. of Electrical Engineering, Princeton University, NJ, USA
[•]Dept. of Information Engineering, The Chinese University of Hong Kong, Hong Kong

***Abstract*—Peer-to-peer (P2P) systems provide a scalable way to stream content to multiple receivers over the Internet. The maximum rate achievable by all receivers is the capacity of a P2P streaming session. We provide a taxonomy of sixteen problem formulations, depending on whether there is a single P2P session or there are multiple concurrent sessions, whether the given topology is a full mesh graph or an arbitrary graph, whether the number of peers a node can have is bounded or not, and whether there are non-receiver relay nodes or not. In each formulation, computing P2P streaming capacity requires the computation of an optimal set of multicast trees, with an exponential complexity, except in three simplest formulations that have been recently solved with polynomial time algorithms. These solutions, however, do not extend to the other more general formulations.**

In this paper, we develop a family of constructive, polynomial-time algorithms that can compute P2P streaming capacity and the associated multicast trees, arbitrarily accurately for seven formulations, to a factor of 4-approximation for two formulations, and to a factor of log of the number of receivers for two formulations. The optimization problem is reformulated in each case so as to convert the combinatorial problem into a linear program with an exponential number of variables. The linear program is then solved using a primal-dual approach. The algorithms combine an outer loop of primal-dual update with an inner loop of smallest price tree construction, driven by the update of dual variables in the outer loop. We show that when the construction of smallest price tree can be carried out arbitrarily accurately in polynomial time, so can the computation of P2P streaming capacity. We also develop several efficient algorithms for smallest price tree construction. Using the developed algorithms, we investigate the impact of several factors on P2P streaming capacity on topologies derived from statistics of uplink capacities of Internet hosts.

I. Introduction

Consider the following problem: given a directed graph with a source node and a set of receivers, how to embed a set of trees spanning the receivers and to determine the amount of flow in each tree, such that the sum of flows over these trees is maximized? Constraints of this problem include an upper bound on the amount of flow from each node to its children, degree of a node in each tree, and other topological constraints of the given graph. This is the basic version of the problem of P2P streaming capacity computation that we try to solve in this paper.

Multicasting content over the Internet can be carried out in two ways: a "client-server" system has one server for each multicast session serving the given set of receivers, and a "peer-assisted" system uses the upload capacity of each user, rather than relying only on the server, to help scale the content delivery as the number of users increases. In a typical P2P system, peering relationships are established among users in the logical overlay network on top of the physical underlay network, giving rise to multiple multicast trees that simultaneously support one session. A user may be in a different level in each of these trees. These P2P systems have enabled scalable file sharing and video streaming since 2001, and consume between one third to half of the entire Internet traffic volume in recent years.

The following fundamental question remains open: what is the P2P streaming capacity and what is an optimal peering configuration to achieve the capacity? Here, capacity is defined as the largest rate that can be achieved for all receivers in a multicast session with a given source, a set of receivers, and possibly a set of helper (non-receiver relay) nodes. Notice that it is not the queuing-theoretic or Shannon-theoretic capacity of a network, as we have assumed infinite backlog for each streaming session and noiseless channels.

There are sixteen formulations of this question: depending on whether there is a single P2P session or there are multiple concurrent sessions, whether the given topology is a full mesh graph or an arbitrary graph, whether the number of peers a node can have is bounded or not, and whether there are helper nodes or not. In each formulation, computing P2P streaming capacity requires the determination of how to embed an optimal set of multicast trees and what should the rate in each tree be. This is generally *exponential complexity*, except in three simplest formulations that have recently been solved with polynomial time combinatorial algorithms [11], [9], [14], [13], [12]. These algorithms and their correctness proofs, however, do not extend to the other formulations that remain open.

In this paper, we develop a family of constructive, *polynomial-time* algorithms that can compute P2P streaming capacity, and the associated multicast trees, arbitrarily accurately for seven formulations, to a factor of 4 approximation for two formulations, and to a factor of log of the number of receivers for two formulations. The optimization is reformulated to turn the combinatorial problems into linear programs with an exponential number of variables. The algorithms combine a primal-dual update outer loop with an inner loop of smallest price tree construction, driven by the update of Lagrange dual variables in the outer loop. Graph-theoretic solutions to various cases of the smallest price tree problem are then presented.

Certain special cases of P2P streaming capacity have been studied recently. For example, [4] develops a primal-dual algorithm for the following special case: undirected overlay graph without degree bounds on nodes or the presence of helper nodes. Existence of degree bounds and helper nodes make the formulations in this paper more relevant to the practice of P2P streaming. They also make the development

V :	Set of all the nodes in the p2p network.
k,K:	Index or total number of sessions.
r,r^k:	Streaming rate for the single, or the k-th session.
s,s^k:	Source of the single, or the k-th session.
R,R^k:	Set of receivers for the single, or the k-th session.
H,H^k:	Set of helpers for the single, or the k-th session.
T,T^k:	Set of all allowed trees on top of $G(V,E)$.
I_t	Set of internal nodes in tree t.
$C(v)$:	Uplink capacity of node $v \in V$.
U_v:	Aggregate uploading rate for node $v \in V$.
x_{uv}:	Streaming rate from node u to node v.
y_t:	Rate of substream delivered by tree t.
$m_{v,t}$:	Outgoing degree of node v in tree t.
$M(v)$:	Bound of the outgoing degree of v in each tree.
$u \rightarrow v$:	u is the parent of v in the tree discussed.
$p_v, p(v)$:	Price, or the dual variable, of node v.
$\mathbf{p}, p(\cdot)$:	Vector or function of prices of all nodes.
$Q(t,p)$:	Price of tree t for a given $p(\cdot)$, or $\mathbf{p}$.

Table I: Main notation.

and analysis of the algorithms more challenging.

The rest of this paper is organized as follows. We introduce the unifying system model in Section II and review related work. We formulate the streaming capacity problem and develop the main algorithms for single-session and multiple-session applications in Sections III, IV, and V. We then present performance evaluation in Section VI. Finally, we conclude in Section VII. All proofs are presented in the appendices.

II. P2P Streaming Model and Capacity Definition

Consider K streaming sessions, indexed by k. A streaming session originates from one source, and is distributed to a given set of receivers. For example, in video conferencing, there are multiple participants, each may initiate a session and distribute her video to others, and each participant can subscribe to others' videos. In an IPTV network, different channels may originate from different servers, with different sets of subscribers. For the k-th session, denote by s^k the original source, by R^k the set of receivers, and by H^k the set of helpers. We define the *rate* r^k of session k if all the receivers in this session receive the streaming packets at or above r^k.

Now consider the P2P network as a graph $G=(V,E)$, where each node $v \in V$ represents a user, and each edge $e=(u,v) \in E$ represents a *neighboring* relationship between vertices (u,v). A user may be the source, or a receiver, or a helper that serves only as a relay. Note that a helper may not need to get all packets. This graph is an overlay on top of the given underlay graph representing the physical connections among users. It may constrain the design of *peering* relationships: if two nodes u and v are not neighbors, they cannot be peered. At the same time, neighbors do not have to become peers. Neighboring relationship is given while peering relationship is to be designed as part of the P2P streaming capacity computation. The graph G may or may not be full mesh. Typically, full mesh is only possible in a small network with small number of users, while a large network has a sparse topology. For example, in P2P systems widely used today, such as BitTorrent or PPLive, when a user joins the system, the server provides a small list of selected users that can be neighbors of the new user and exchange packets among them.

Consider a given stream and a packet in it: it starts from the source s^k, and traverses over all nodes in R^k, and some nodes in H^k – the traversed paths form a Steiner tree in the overlay graph $G(V,E)$. Different packets may traverse different trees, and we call each tree a sub-stream tree, or *sub-tree* or simply tree in short, and call the superposition of all the sub-trees belonging to the same session a *multi-tree*. For each tree $t \in T^k$, we denote by y_t the rate of the sub-stream supported by this tree.

There are P2P protocol constraints on sub-trees. The most frequently encountered one is degree constraint. For example, in BitTorrent, although one node has $30-50$ neighbors in G, it can upload to at most 5 of them as peers. This gives an outgoing degree bound for each node and constrains the construction of the trees. Degree bounds always apply to receivers and helpers, and, in some scenarios, to the source as well. Let $m_{v,t}$ be the number of outgoing edges of node v in tree t, and the bound be $M(v)$: $m_{v,t} \le M(v), \forall t$. We denote by T^k the set of all allowed sub-trees for the k-th session: trees that satisfy the constraints such as the degree bounds. Obviously, rate $r^k = \sum_{t \in T^k} y_t$ for all k.

We will make the following assumptions for streaming applications: there is a static set of stationary users and all desired chunks of packets are available at each node. The issues of peer churn and chunk availability will be studied in future work. We also assume that data rate bottlenecks only appear at user uplinks. This assumption is widely adopted in the P2P literature because in today's Internet, access links are the bottlenecks rather than backbone links, and uplink capacity is several times smaller than downlink capacity in access networks. Denote by $C(v)$ the uplink capacity of node v. We have the following bound on the total uplink rate U_v for each node v:

$$U_v := \sum_{u \in V} x_{vu} \le C(v)$$

where x_{vu} is the streaming rate node v transmits to node u.

A rate is called *achievable* if there is a multi-tree in which all trees satisfy the topology constraint ($t \in T^k$) and transmission rates satisfy the uplink capacity constraint ($U_v \le C(v)$). We define *P2P streaming capacity* C as the largest achievable rate. When there are multiple sessions in the same given graph G, each session's capacity is denoted as C^k, the P2P streaming rate region is defined as the set of $\{C^k\}$ that can be simultaneously achieved, and the *P2P streaming capacity region* is the Pareto-optimal boundary of the rate region. Except in relatively easy special cases, these fundamental limits of P2P performance are unknown. The rest of this paper studies the polynomial-time computation of P2P streaming capacity or capacity region, and the multi-trees that achieve them.

A taxonomy of P2P streaming capacity is shown as a "tree of problem formulations" in Figure1. Each leaf node in this tree is a specific problem formulation. There are four levels of dichotomy: whether is there is one session or multiple sessions, whether G is full mesh or not, whether there are degree bounds for each node in the trees or not, and whether there are helper nodes or not. Some formulations are significantly more difficult than others. When G is not full mesh, not all nodes can become neighbors in a tree. When there are degree bounds per node per tree, the constraint set T^k further complicates

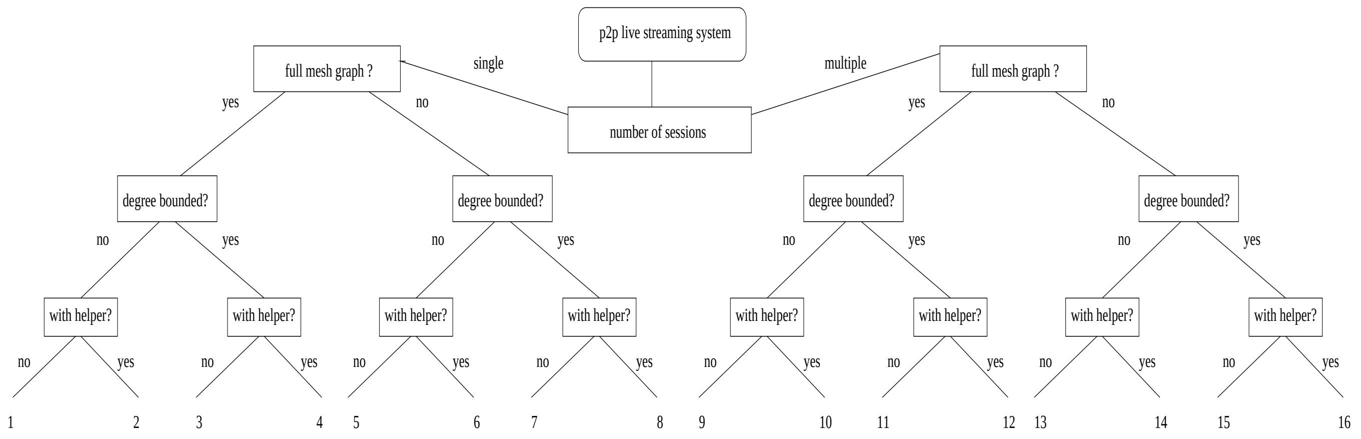

Fig. 1. Classification of P2P streaming capacity problems: single or multiple session, full mesh or non-full mesh graph G given, bounded or unbounded node degrees in each tree, and existence or absence of helper nodes. The combination of these four classifications leads to 16 cases of problem formulations.

	Small Network	Large Network
IPTV	1,2	3,4,5,6,7,8
Video Conferencing	9,10	11,12,13,14,15,16

Table II: The connections between typical application scenarios and problem formulations (Problems 1-16 in Figure 1). IPTV is a single session for each channel, while video conferencing typically contain multiple sessions. Full mesh graph without degree bound models a small network, while either non-full mesh graph or degree bounded tree models a large network.

Formulation	Accuracy	Location
1,2	$1+\varepsilon$	Sec III, V-A
3,4	$1+\varepsilon$	Sec III, V-B
5	$1+\varepsilon$	Sec III,V-C
6	$\ln\|R\|\ln^3\|V\|+\varepsilon$	Sec III,V-C
7	$4+\varepsilon$	Sec III,V-D
8	open	Sec III,V-D
9,10	$1+\varepsilon$	Sec IV, V-A
11,12	$1+\varepsilon$	Sec IV, V-B
13	$1+\varepsilon$	Sec IV, V-C
14	$\ln\|R\|\ln^3\|V\|+\varepsilon$	Sec IV, V-C
15	$4+\varepsilon$	Sec IV, V-D
16	open	Sec IV, V-D

Table III. Summary of the results in this paper for the sixteen problem formulations.

the tree optimization. When there are helper nodes, capacity may be increased but the task of computing capacity also becomes more challenging. In formulations 1 and 2, capacity can be computed exactly and in polynomial time as shown in recent papers using combinatorial algorithms [11]. The other formulations are much more difficult, and form the subject of study in this paper. We also summarize in Table II the connection between typical applications and our classification, and in Table III, the state of solution for each problem formulation in this paper.

III. Single Session

We first consider the single session case. Typical applications include the streaming of a single channel in IPTV and single-source video conferencing. In particular, the source of IPTV is usually a powerful server and it is either not degree bounded, or has a very large bound. The source of video conference is usually a peer node, and its degree bound is of the same order as the other peers.

Since there is only one multicast session, we remove the superscripts k, and denote by s the sender, by R the set of receivers, by H the set of helpers, by T the set of all allowed sub-trees, and by r the supported streaming rate in the multi-tree. We further use $|R|$ and $|H|$ to denote the number of receivers and helpers, respectively. The total number of users in the system is $N=1+|R|+|H|$.

A. Problem Formulation

We represent the single-session streaming capacity problem as the following optimization, where the objective function and constraints are as explained in the previous section. For those trees not selected in the optimizer, their rates y_t are simply 0. The representation may be deceptively simple: the difficulty lies in searching through all combinations of trees t in the set of allowed trees T.

Single-Session (Primal) Problem

$$\begin{array}{lll} \text{maximize} & r=\sum_{t\in T} y_t & (1)\\ \text{subject to} & \sum_{t\in T} m_{v,t} y_t \le C(v)\,,\ \forall\ v\in V & (2)\\ & y_t\ge 0\ \forall\ t\in T & (3)\\ \text{variables} & y_t, \forall t\in T & (4) \end{array}$$

From linear programming duality theory [1], solving the above problem is equivalent to solving its dual problem, and an optimizer of the dual problem readily leads to an optimizer of the primal algorithm. The dual problem associates a non-negative variable $p(v)$, interpreted as price, with each node v corresponding to constraint (2). It can be derived to be the following problem:

Single-Session Dual Problem

$$\begin{array}{lll} \text{minimize} & \sum_{v\in V} C(v)p(v) & (5)\\ \text{subject to} & \sum_{v\in V} m_{v,t} p(v)\ge 1\,,\ \forall\ t\in T, & (6)\\ & p(v)\ge 0\ \forall\ v\in V & (7)\\ \text{variables} & p(v), \forall v\in V & (8) \end{array}$$

We can interpret the dual problem this way: $p(v)$ is the per unit flow price for any edge outgoing from v. If node v uploads with full capacity, the incurred cost is $p(v)C(v)$. There are $m_{v,t}$ connections outgoing from node v in tree t, and thus the *total tree price* for tree t, which is defined as the sum of prices in any edge in tree t, is $\sum_{v\in V} m_{v,t}p(v)$. Therefore, the dual problem is to minimize the total full capacity tree cost given that the tree price is at least 1, and the minimization is over all possible $\mathbf{p}$, where $\mathbf{p} := \{p(v), \forall v \in V\}$ is the price vector. For notational simplicity, we use $p(\cdot) : V \to R^+ \bigcup\{0\}$ to represent $\mathbf{p}$.

In general, the number of trees we need to search when computing the right multi-tree grows exponentially in the size of the network. This dimensionality increase is the consequence of turning a difficult graph-theoretic, discrete problem into a continuous optimization problem. Hence, the primal problem can have possibly exponential number of variables and its dual can have an exponential number of constraints, neither of which suitable for direct solution if we want to compute P2P streaming capacity in polynomial time as the network size increases. However, the above representations turn out to be very useful to allow a primal-dual update outer loop that converts the combinatorial problem of multi-tree construction into a much simpler problem of smallest price tree construction.

B. Algorithm and Performance

We now design an iterative combinatorial algorithm that solves the primal and dual problems approximately. We adapt the technique for solving the maximum multi-commodity flow problem in [6], where flows are augmented in the primal solution and dual variables are updated iteratively. Our algorithm constructs peering multi-trees that achieve an objective function value within $(1+\zeta)$-factor of optimal.

For a given tree t and prices $p(\cdot)$, let $Q(t,p)$ denote the left-hand-side (LHS) of constraint (6), which we call the *price* of tree t. A set of prices $p(\cdot)$ is a feasible solution for the dual program if and only if

$$\min_{t\in T} Q(t,p) \geq 1.$$

The algorithm works as follows. Start with initial weights $p(v) = \frac{\delta}{C(v)}$ for all $v \in V$. Parameter δ depends on ζ and is described in more detail later. Repeat the following steps until the dual objective function value becomes greater than 1:

1) Compute a tree $\bar{t}$ for which $Q(t,p)$ is minimum. We call $\bar{t}$ a *smallest price tree* problem, algorithms for which are developed in Section V.
2) Send the maximum flow on this tree $\bar{t}$ such that uplink capacity of at least one internal node is saturated. Let $I(t)$ be the set of internal nodes in tree t. The flow sent on this tree is
$$y = \min_{v\in I(\bar{t})} \frac{C(v)}{m_{v,\bar{t}}}. \qquad (9)$$
3) Update the prices $p(v)$ as
$$p(v) \leftarrow p(v)\left(1+\frac{\varepsilon m_{v,\bar{t}} y}{C(v)}\right), \ \forall\ v \in I(\bar{t}).$$
where ε depends on θ and is explained in more detail later.
4) Increment the flow Y sent so far by y.

The optimality gap can be estimated by computing the ratio of the primal and dual objective function values in each step of the above iteration, which can be terminated after the desired proximity to optimality is achieved. When the above iteration terminates, primal capacity constraints on each uplink may be violated, since we were working with the original (and not residual) uplink capacities at each stage. To remedy this, we scale down the flows uniformly so that uplink capacity constraints are satisfied.

The pseudo-code for the above procedure is provided in Figure 2. Array $flow(v)$ keeps track of the traffic on uplink of node v as the algorithm progresses. The dual objective function value is tracked by variable D which is initialized to 0. After the "while" loop terminates, the maximum factor by which the uplink capacity constraint is violated on any uplink is computed as α, which divides the total flow Y, and the resulting value is output as r^*.

```
Primal-Dual Algorithm: Single-Session
$p(v) \leftarrow \frac{\delta}{C(v)}, flow(v) \leftarrow 0,\ \forall\ v \in V,\ Y \leftarrow 0,$
$D \leftarrow 0$

while $D < 1$
    Pick tree $t \in T$ with the smallest $Q(t,p)$
    $y \leftarrow \min_{v\in I(t)} C(v)/m_{v,t}$
    $\bar{t} \leftarrow \arg\min_{v\in I(t)} C(v)/m_{v,t}$
    $flow(v) \leftarrow flow(v) + y m_{v,\bar{t}}, \forall v \in I(\bar{t})$
    $Y \leftarrow Y + y$
    $p(v) \leftarrow p(v)(1+\varepsilon \frac{m_{v,\bar{t}} y}{C(v)})$
    $D \leftarrow \sum_{v\in V} C(v)p(v)$
end while

Compute scaling factor $\alpha \leftarrow \max_{v\in V} \frac{flow(v)}{C(v)}$ ;
Output capacity $r^* \leftarrow Y/\alpha$ ;
```

Fig. 2. The Primal-Dual Algorithm for Single-session P2P Streaming Capacity Computation.

The following theorem, proved in Appendix A, states accuracy and complexity properties of the algorithm:

Theorem 1. *For any given $\zeta > 0$, the Single-Session Primal-Dual Algorithm computes a solution with objective function value within $(1+\zeta)$-factor of the optimum, for algorithmic parameters $\varepsilon(\zeta) = 1 - \frac{1}{\sqrt{1+\zeta}}$ and $\delta(\zeta) = \frac{1+\varepsilon}{[(1+\varepsilon)|V|]^{1/\varepsilon}}$. It runs in time polynomial in the input size and $\frac{1}{\varepsilon}$: $O\left(\frac{|V|\log|V|}{\varepsilon^2}T_{spt}\right)$, where T_{spt} is the time to compute a smallest price tree.*

This unifying primal-dual framework works for *all* of the single session problems in Section II. The core issue now lies with the inner loop of smallest price tree computation: can this be accomplished in polynomial time for a given price vector? This graph-theoretic problem is generally more tractable than the original problem of searching for the multi-tree that maximizes the achievable rate. However, when the given graph G is not full mesh, or when there are degree bounds on nodes in each tree, or when there are helper nodes, computing a smallest price tree becomes difficult. These are described in Section V which is devoted to constructing smallest price trees for the various cases.

IV. Multiple Sessions

Now we turn to the more general case of K concurrent streaming sessions sharing the same P2P network. Each session is supported by a multi-tree. This models the multi-channel IPTV and multi-party video conference scenarios. Here the notion of capacity becomes a region in the K-dimensional space, with tradeoffs among the sessions quantified by the shape of this capacity region.

A. Problem Formulation

Given a session rate demand vector $[r^1, r^2, \cdots, r^K]$, let λ be the maximum multiplier such that session rate λr^k can be supported for session k. Hence, we have

$$\sum_{t \in T^k} y_t = \lambda r^k, \;\; k = 1, 2, \cdots, K.$$

The total uplink traffic at node v is $\sum_k \sum_{t \in T^k} m_{v,t} y_t$. Upload link capacity constraint becomes

$$\sum_k \sum_{t \in T^k} m_{v,t} y_t \le C(v), \;\; \forall \; v \in V.$$

For a given "rate region direction vector" $\{r^k\}$, solving the following problem provides one point on the capacity region. By varying $\{r^k\}$, all points can be traced. This is the same as scalarization of a vector-valued optimization problem and sweeping through the scalarization parameter to obtain the entire boundary of the tradeoff region, a polyhedron in this case.

Multi-Session (Primal) Problem

$$\begin{array}{lll}
\text{maximize} & \lambda & (10) \\
\text{subject to} & \sum_{t \in T^k} y_t = \lambda r^k, \;\; k = 1, 2, \cdots, K & (11) \\
 & \sum_k \sum_{t \in T^k} m_{v,t} y_t \le C(v), \;\; \forall \; v \in V & (12) \\
 & y_t \ge 0, \; \forall \; t \in T^k, \;\; k = 1, 2, \cdots, K & (13) \\
\text{variables} & \lambda, y_t, \forall t \in T^k, \;\; , k = 1, 2, \cdots, K & (14)
\end{array}$$

In the special cases of formulations 1 and 2 in Figure 1, the set of trees T^k for session k can be chosen to comprise of a linear of Mutualcast trees [11] and this guarantees the solution to be optimal. Hence, the linear program is of polynomial size and can be solved in polynomial time in these cases.

We can also readily derive the dual linear programming problem for the multi-session case. The dual problem associates a variable z_k with each session k corresponding to constraint (11), and a non-negative variable $p(v)$ with each node v corresponding to constraint (12).

Multi-Session Dual Problem

$$\begin{array}{lll}
\text{minimize} & \sum_{v \in V} C(v) p(v) & (15) \\
\text{subject to} & \sum_{v \in V} m_{v,t} p(v) \ge z_k, \; \forall \; t \in T_k, \; \forall \; k = 1, \cdots, K & (16) \\
 & \sum_{k=1}^{K} r^k z_k \ge 1 & (17) \\
 & p(v) \ge 0 \; \forall \; v \in V & (18) \\
\text{variables} & p(v), \forall v \in V, z_k \forall k = 1, \cdots, K & (19)
\end{array}$$

B. Algorithm and Performance

For multi-session P2P streaming capacity region computation, we again construct an outer loop of primal-dual update that is complemented by an inner loop of smallest price tree construction. A key observation is that, from constraint (16), for each session k, z_k can be set to be the minimum value of left-hand-side, over all $t \in T^k$, under given prices $p(v)$ for all $v \in V$. Hence, a set of prices $p(v)$ is a dual feasible solution if constraint (17) is satisfied, after z_k is determined. If constraint (17) is not satisfied for a given set of prices $p(v)$, these prices can be scaled to satisfy the constraint.

Start with initial prices $p(v) = \frac{\delta}{C(v)}$ for all $v \in V$. The algorithm proceeds in phases. In each phase, we route r^k units of flow from node s^k to receivers in R^k along the multi-trees in T^k, for each session k. A phase ends when all sessions $k = 1, 2, \cdots, K$ have been routed.

The flow of value r^k of session k is routed from s^k to receivers in R^k in multiple iterations. In each iteration, a tree $t \in T^k$ that minimizes the LHS of constraint (16) under current prices $p(v)$ is computed. The maximum flow u that can be sent on this tree t subject to *original* node uplink constraints is given by

$$u = \min_{v \in V, m_{v,t} > 0} \frac{C(v)}{m_{v,t}}.$$

The amount of flow sent along tree t, denoted by y_t, in an iteration is the minimum of (i) the quantity u, and (ii) the remaining amount of flow that needs to be sent from s^k to receivers in R^k to make a total of r^k.

After the flow of value y_t is sent along tree t, the prices $p(v)$ and the uplink flow values at each node are updated as follows:

1) Update the prices $p(v)$ as

$$p(v) \leftarrow p(v) \left(1 + \frac{\varepsilon m_{v,t} y_t}{C(v)} \right), \;\; \forall \; v \in V.$$

2) Increment the uplink flow value for each node v by $m_{v,t} y_t$.

This update happens after each iteration associated with routing a *portion* of flow r^k for each session k. The algorithm terminates when the dual objective function value becomes less than unity.

When the algorithm terminates, dual feasibility constraints will be satisfied. However, link capacity constraints (12) in the primal solution will be violated, since we were working with the original (not the residual) uplink capacities at each stage. To remedy this, we scale down the traffic at each node uniformly so that uplink capacity constraints are satisfied.

Again, we need smallest price tree algorithms in the next section to compute a tree $t \in T^k$ that minimizes the LHS of constraint (16) during each iteration.

```
Primal-Dual Algorithm: Multi-Session
$p(v) \leftarrow \frac{\delta}{C(v)}, flow(v) \leftarrow 0, \forall v \in V, phase \leftarrow 0$

repeat
    for each $k = 1, 2, \cdots, K$ do
        $r = r^k$ ;
        while $r > 0$
            Compute minimum cost tree $t \in T^k$ under
            prices $p(v)$ so as to minimize $\sum_{v \in V} m_{v,t} p(v)$ ;
            $u \leftarrow \min_{v \in V, m_{v,t} > 0} \frac{C(v)}{m_{v,t}}$ ;
            $y_t = \min(r, u)$ ;
            for each $v \in V$ do
                $p(v) \leftarrow p(v) \left(1 + \frac{\varepsilon m_{v,t} y_t}{C(v)}\right)$ ;
                $flow(v) \leftarrow flow(v) + m_{v,t} y_t$ ;
            end for
            $r \leftarrow r - y_t$ ;
        end while
    end for
    $phase \leftarrow phase + 1$ ;
    $D \leftarrow \sum_{v \in V} C(v) p(v)$ ;
until $D \geq 1$ ;

Compute $\mathfrak{L}(v) \leftarrow \max_{v \in V} \frac{flow(v)}{C(v)}$,
Computer scaling factor $\mathfrak{L} \leftarrow \max_{v \in V} scale(v)$ ;
Output $\lambda = phase/\mathfrak{L}$ ;
```

Fig. 3. The Primal-Dual Algorithm for Multi-Session P2P Streaming Capacity Computation.

The pseudo-code for the above procedure is described in Figure 3. Array $flow(v)$ keeps track of the uplink traffic at node v. After the completion of a phase, the variable *phase* equals the number of phases completed. The dual objective function value is computed as D at the end of each phase. The iteration over the phases in the "repeat" loop continues as long as this value remains less than one. After the "repeat" loop terminates, the maximum factor by which the uplink capacity constraint at a node v gets violated is computed into variable $\mathfrak{L}$. Finally, the value of λ is output.

Similar to the single-session case, the multi-session primal-dual algorithm also achieves an objective function value within $(1+\zeta)$-factor of optimal, as stated in the following theorem proved in Appendix B.

Theorem 2. *For any $\zeta > 0$, the Multi-Session Primal-Dual Algorithm computes a solution with objective function value within $(1+\zeta)$-factor of the optimum, if the algorithmic parameters are $\varepsilon(\zeta) = 1 - (1-\zeta)^{1/3}$ and $\delta(\zeta) = \left(\frac{1-\varepsilon}{|V|}\right)^{1/\varepsilon}$. It runs in time $O\left(\frac{K \log K}{\varepsilon} \log_{1+\varepsilon} \frac{|V|}{1-\varepsilon} T_{spt}\right)$, where T_{spt} is the time to compute a smallest price tree.*

V. Computing Smallest Price Tree

For iterative algorithms for both the single and multiple session cases, efficiently and accurately computing a smallest price tree (SPT) for a given set of node prices is the key module in each outer loop. Theorems 1 and 2 can also be readily extended to show that, if there is an α-approximation algorithm ($\alpha \geq 1$) for the SPT problem, then the primal-dual algorithm can guarantee an approximation factor of $1/(\alpha + \zeta)$ for any $\zeta > 0$. We now develop SPT algorithms for increasingly more general problem formulations.

A. Full Mesh Graph without Degree Bound

We start with the simplest case: the given graph G is full mesh (all nodes are neighbors of each other), and there is no degree bound in the trees (each node can form peering relationship with any number of other nodes).

When there is no helper node, it is easy to construct the smallest price tree t^* in the following way. Given $p(\cdot)$, let $v^* \in \mathrm{argmin}_{v \in V} p(v)$ be the node with the smallest price. If there are multiple such nodes, we can randomly pick one. If $v^* = s$, let t^* be a 1-hop tree: $s \rightarrow v, \forall v \in R$. Otherwise, let t^* be a 2 hop tree: $s \rightarrow v^* \rightarrow v, \forall v \neq v^* \in R$. The resulting tree is a smallest price tree with $p(t^*) = p(s) + (|R|-1)p(v^*)$.

The presence of helpers complicate the SPT computation. We define an *effective price* for all the nodes in the following way:

$$\hat{p}(v) = \begin{cases} p(v) & \text{if } v \in s \bigcup R \\ p(v)\frac{|R|}{|R|-1} & \text{if } v \in H. \end{cases} \tag{20}$$

With the effective prices, it turns out that we can treat helpers the same way as receivers, and have the following algorithm for both the cases with or without helpers, as described in Figure 4.

```
SPT Computation: Full Mesh Graph, No Degree Bound

Pick $v^* \in \mathrm{argmin}_{v \in V} \hat{p}(v)$
    If $v^* = s$, construct $t^* : s \rightarrow v, \forall v \in R$.
    If $v^* \in R$, construct $t^* : s \rightarrow v^*, v^* \rightarrow v, \forall v \neq v^* \in R$.
    If $v^* \in H$, construct $t^* : s \rightarrow v^*, v^* \rightarrow v, \forall v \in R$.
$t^*$ is the smallest price tree.
```

Fig. 4. The SPT computation module: Full mesh graph G, and no degree bound in multicast trees.

The following theorem is proved in Appendix C:

Theorem 3. *For a full mesh graph with no degree bound on trees, the algorithm in Figure 4 computes an SPT optimally in linear time.*

B. Full Mesh Graph with Degree Bound

We now study a more complicated case, where the given graph is still full mesh, but the outgoing degree in each tree in the P2P design is bounded. Define $M(v)$ the outgoing degree bound for node v, then $m_{v,t} \leq M(v), \forall t \in T$.

With degree bounds, we cannot simply find the smallest price node and forward all receivers from that node. Instead, we need more than two internal nodes, and we want the internal nodes to have as small prices as possible. We make I smallest price receivers be the internal node, and let these small price nodes forward with maximum degrees until all receivers are included in the tree. For the R receivers, we order them by their capacity so that

$$C(1) \le C(2) \le \cdots \le C(|R|).$$

For a given positive integer n, define

$$I(n,M) = \min\{i \in R : \sum_{v=1}^{i} M(v) \ge n\}. \qquad (21)$$

Receivers 1 to $I(n,M)$ can support n nodes altogether. If source s has $m(s)$ children, it is obvious that the smallest price tree with source degree $m(s)$, denoted by $t_{m(s)}$, satisfies the following properties:

$$m(v) = \begin{cases} m(s), & \text{if } v = s \\ M(v), & \text{if } 1 \le v < I(|R|-m(s),M \\ m_I, & \text{if } v = I(|R|-m(s),M) \\ 0, & \text{if } v > I(|R|-m(s),M) \end{cases}$$

where $m_I := |R| - m(s) - \sum_{j=1}^{I(|R|-m(s),M)-1} M(j)$. This means that, in tree $t_{m(s)}$, node 1 to $I(|R|-m(s),M)-1$ have maximum degrees and the last internal node $I(|R|-m(s),M)$ take the remaining receivers as children. The smallest price tree is thus the minimum price tree among $t_{m(s)}$ for all $m(s) : 1 \le m(s) \le M(s)$, and

$$Q(t^*,p) = \min_{m=1}^{M(s)} Q(t_m,p).$$

Based on the above argument, we design the algorithm in Figure 5 for the case without helper nodes.

```
SPT Computation: Full Mesh Graph, Degree Bound, No Helper

Pick $v \in R$ with the smallest price
$A = \{s,v\}, B = R - \{v\}$
$s \to v$,  $m(s) = 1$,  $m(v) = 0, \forall v \in R$

while $A \neq V$
    $\tilde{A} = \{v \in A : m(v) < M(v)\}$
    Pick $v_a \in \tilde{A}$ with the smallest price
    $m' = \min(|B|, M(v_a) - m(v_a))$
    Find $m'$ smallest price nodes in $B$, denote the set by $D$
    $A = A \bigcup D$,  $B = B - D$
    $v_a \to v, \forall v \in D$,  $m(v_a) = m(v_a) + m'$
end while
```

Fig. 5. The SPT computation module: Full mesh graph G with degree bounds on nodes in the multicast trees and no helper nodes.

The following result is proved in Appendix D:

Theorem 4. *For a full mesh graph with degree bound on trees, the algorithm in Figure 5 computes an SPT optimally in linear time.*

We now allow the presence of helper nodes, and study a special case for this scenario: $M(v) = M_p, \forall v \in R \bigcup H$, and $M(s) = \infty$. From Subsection V-A, we have already seen that, if a tree contains a helper, it contains more edges than the helper-free tree. Therefore, if $p(v_h) < p(v_r), v_h \in H, v_r \in R$, it does not necessarily mean that v_h is more favored to be a parent than v_r. To compare a helper with a receiver, we need to define some "effective prices" under the degree bound $M(v) = M_p, \forall v \in R \bigcup H$.

```
SPT Finding: Full Mesh, Degree Bound, with Helpers

$m = \min(M_p, |R|)$
$\hat{p}(v) = p(v)(1 + \frac{1}{m-1}), \forall v \in H$, $\hat{p}(v) = p(v), \forall v \in R \bigcup s$
Pick $v \in R \bigcup H$ with the smallest effective price $\hat{p}(v)$
$A = \{s,v\}, B = R \bigcup H - \{v\}$
$s \to v$,  $m(s) = 1$,  $m(v) = 0, \forall v \in R$

while $B \bigcap R \neq \emptyset$
    $n = \sum_{v \in A, v \neq s} (M(v) - m(v))$
    if $n \ge |B \bigcap R|$
        break
    end if
    $m = \min(M_p, |B \bigcap R| - n)$, $\hat{p}(v) = p(v)(1 + \frac{1}{m-1}), \forall v \in H$
    Pick $u \in B$ with the smallest effective price
    if $\hat{P}(u) \ge p(s)$
        break
    end if
    $\tilde{A} = \{v \in A : m(v) < M(v)\}$
    Pick $v_a \in \tilde{A}$ with the smallest effective price
    $v_a \to u$, $m(v_a) = m(v_a) + 1$, $A = A \bigcup u$, $B = B - u$.
end while

$B = B \bigcap R$
while $B \neq \emptyset$
    $\tilde{A} = \{v \in A : m(v) < M(v)\}$
    Pick $v_a \in \tilde{A}$ with the smallest price
    $m' = \min(|B|, M(v_a) - m(v_a))$
    Find $m'$ smallest price nodes in $B$, denote the set by $D$
    $A = A \bigcup D$,  $B = B - D$
    $v_a \to v, \forall v \in D$,  $m(v_a) = m(v_a) + m'$
end while
```

Fig. 6. The SPT Finding Module. Full mesh graph with homogeneous non-source peer degree bound, with helpers.

Consider a tree t containing a helper v_h. A leaf helper is meaningless, and we suppose v_h has $m \le M_p$ children. If we wish to remove this helper, we can replace v_h with a leaf node v_r, and let v_r support the children of v_h. By replacing v_h with v_r, the degree of v_r's parent node is decreased by 1, and we can thus let v_r support $m-1$ children and let v_r's old parent node support the last child. Then we arrive at a new tree t' where

$$Q(t',p) - Q(t,p) = (m-1)p(v_r) - mp(v_h).$$

Therefore, we should compare $mp(v_h)/(m-1)$ with $p(v_r)$ when deciding who should be the parent. Therefore, we redefine the effective price for v_h to be $\hat{p}(v_h) = (1 + 1/(m-1))p(v_h)$ if v_h is to support m children. Only when the effective price of a helper is smaller than the price of receiver v_r, v_h is more favored than v_r to be a parent.

Based on the above analysis, we modify the helper-free algorithm in Figure 5 and arrive at a linear-complexity SPT computation algorithm in Figure V-B.

C. General Graph without Degree Bound

We now study the cases where the given graph G is not full mesh, thus for any given node, not all other nodes are its neighbors that can be peered. First are the cases without degree bounds in the trees. For the helper-free case, the SPT computation problem becomes the minimum cost arborescence problem (rooted directed spanning tree), which is has been solved in [3]. Hence, we consider the case with the presence of helpers. In this case, the SPT computation problem is a minimum cost directed Steiner tree problem *with symmetric connectivity and a special structure on the costs* – costs of all edges going out of a node are equal. We leverage these two special features to accelerate the algorithm through a graph transformation.

Let $[v_1, v_2]$ represent the pair of links connecting two neighboring nodes v_1 and v_2. We want to find the minimum price directed Steiner tree connecting source s and a set of receivers in R. We transform the directed graph G to an undirected graph $G' = (V', E')$, which represents the adjacency between link pairs and the source node s:

- For source node s, we copy it into G'.
- For every two neighboring nodes $v_1, v_2 \in V$, we map the link pair $[v_1, v_2]$ to a node $n_{[v_1,v_2]}$ in G'.
- For every node $v_1 \in V$ in G, we map it to a series of *undirected* links connecting nodes $n_{[v_1,v_2]}$ and $n_{[v_1,v_3]}$ in G' where v_2 and v_3 are any neighbors of v_1 in G; we set the prices of these undirected links to $p(v)$.
- In graph G', we connect any two of nodes s and $n_{[s,v]}$, with a series of *undirected* links, where v is any neighbor of s in G; we set the prices of these links to be $p(s)$.

An example of such transformation is illustrated in Fig. 7.

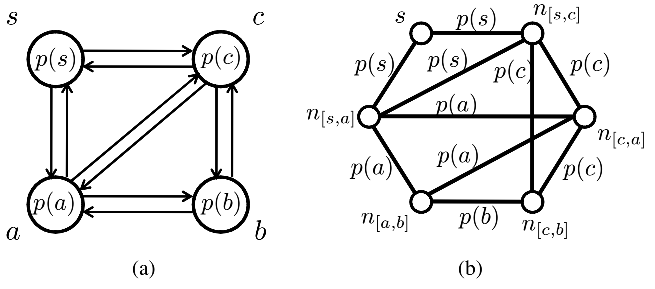

Fig. 7. a) An overlay graph with source node s, receiver nodes a and b, and Steiner node c with node prices being p_s, p_a, p_b, and p_c respectively; b) The undirected graph mapped from the overlay graph in a); for example, the link pair between a and b in a) maps to the node $n_{[a,b]}$ in b), and node a maps to three links with link cost p_a in b).

For the undirected graph G', we consider the following group Steiner tree problem. For every receiver $r \in R$ in G, we group all the nodes $n_{[r,v]} \in V'$, $v \in V$, into a set, denoted by g_r. Set g_r in G' corresponds to the set of link pairs in G connecting r to all its neighbors. We also construct a set g_s that contains only the source s in G'. The group Steiner tree problem in G' is to find the minimum price Steiner tree that connects at least one node from each of sets g_s and g_r, $r \in R$. Solving the group Steiner tree problem in undirected graph is NP-hard [7]. The authors in [7] propose a polynomial time algorithm that achieves an approximation factor of $\frac{1}{O(\ln N_g \ln^3 N_m)}$, where N_g is the number of groups and N_m is total number of nodes.

The following theorem, proved in the appendix, states that finding the minimum cost Steiner tree in G is equivalent to searching the minimum cost group Steiner tree in G'.

Theorem 5. *Consider finding a Steiner tree in G that connects s and all nodes in R, and searching a group Steiner tree in G' that connects at least one node from each of node sets g_s and g_r, $r \in R$, the followings are true:*

1) *a directed Steiner tree in G can be mapped to a group Steiner tree in G' with the same price in polynomial time.*
2) *a group Steiner tree in G' can be mapped to a directed Steiner tree in G with the same or less price in polynomial time.*

Consequently, the optimal group Steiner tree in G' can be mapped to the optimal directed Steiner tree in G in polynomial time and vice versa. Furthermore, their prices are equal.

The minimum cost directed Steiner tree problem is hard to approximate to a factor better than $\frac{1}{\ln|R|}$ [5]. An $\frac{1}{O(|R|^{\varepsilon})}$-factor approximation algorithm that runs in polynomial time for any fixed $\varepsilon > 0$ is given in [2]. Theorem 5 states that the directed Steiner tree problem in graph G can be approached by studying a group Steiner tree problem in G'. We first apply the randomized algorithm proposed in [7] to G', and get a group Steiner tree with an approximation factor of $\frac{1}{O(\ln|R| \ln^3 N_m)}$. N_m corresponds to total number of link pairs in G, and is at most $|V|^2$. We then map this group Steiner tree to a directed Steiner tree in G. Since this mapping keeps or reduces the price, at the end we compute a directed Steiner tree in G with an approximation factor of $\frac{1}{O(\ln|R| \ln^3|V|)}$ in polynomial time. In the case where $|V| = O(|R|)$, we can compute a directed Steiner tree with an approximation factor of $\frac{1}{O(\ln^4|R|)}$ in polynomial time.

D. General Graph with Degree Bound

The most general cases of SPT computation are much harder than all cases before: the given graph G is not full mesh and there are degree bounds in the multicast trees. Even determination of the existence of a feasible tree is NP-hard. When there are no helpers, the problem can be solved as a special case of a factor-4 approximation recently developed in [10]. It remains an open problem on whether this approximation can be further improved by leveraging the special structure in SPT computation that the prices of all the links going out of a node are equal. When there are helpers, polynomial-time computation of SPT for any factor of approximation accuracy is completely open.

E. Summary

As illustrated in Figure 8, before this paper the approach towards computing P2P streaming capacity was entirely combinatorial and successful only for the simple cases such as formulations 1, 2, and 3. Now we have added an alternative approach: an outer-loop of primal-dual update that provides

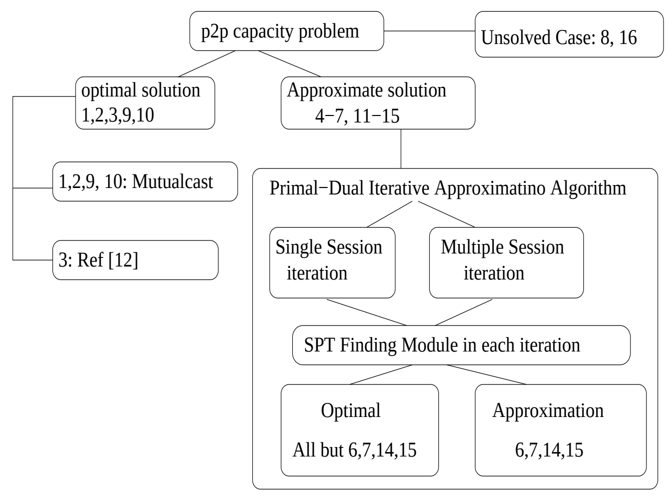
p2p capacity problem
Unsolved Case: 8, 16
optimal solution 1,2,3,9,10
Approximate solution 4−7, 11−15
1,2,9, 10: Mutualcast
3: Ref [12]
Primal−Dual Iterative Approximatino Algorithm
Single Session iteration
Multiple Session iteration
SPT Finding Module in each iteration
Optimal All but 6,7,14,15
Approximation 6,7,14,15

Fig. 8. Schematic for algorithms for the P2P streaming capacity problem. *All approximation algorithms are contributions of this paper*.

pricing guidance on how much more can a node be of help to distribute the content, embedding an inner-loop of the less challenging, though sometimes still difficult, combinatorial problem of SPT computation for the given prices. As long as SPT computation can be carried out accurately and efficiently, P2P streaming capacity can be computed accurately and efficiently as in Theorems 1 and 2. Polynomial time computation of SPT is now shown as the case for 10 of the 16 formulations of the P2P streaming capacity problem.

VI. Evaluation

The algorithms developed in this paper can be used either as an offline benchmarking tool or embodied in the control plane of P2P streaming systems to construct peering relationships that achieves the capacity. In our numerical simulations, we consider networks with $n = 10, 100, 1000, 10000$ nodes and draw the node uplink capacities from a distribution that is obtained from real peer usage data as reported in [8]. The possible uplink capacities of peers and their respective fractions in the peer population is summarized in Table VI.

Uplink Capacity (Kbps)	64	128	256	384	768
Fraction (%)	2.8	14.3	4.3	23.3	55.3

Table IV: Peer Uplink Capacity Distribution

A. Algorithm Accuracy

We first examine a scenario with a full-mesh G, degree bound $M = 2$, and no helpers. We compare the streaming rates computed by two different algorithms – the primal-dual (and SPT) based algorithm in Sections 3 and 4, which is applicable for the general case, and the mutualcast algorithm in [11], which only works optimally for the case of full-mesh G without degree bounds. It turns out that in this numerical example $M = 2$ is sufficiently loose that it becomes equivalent to the case of no degree bound. We plot the r_{max} vs. $C(s)$ curve for each for a range of source uplink capacities $C(s)$ in Figure 9. This experimentally validates the $1/(1+\theta)$-factor approximation optimality of the primal-dual algorithm, where in this case we set $\theta = 10\%$.

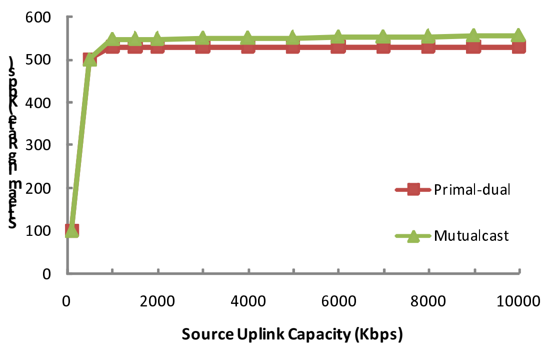

Fig. 9. Streaming Rate for different algorithms (for IPTV case) as a function of source uplink capacity; $n = 1000$ nodes, degree bound $M = 2$.

As the source capacity increases, the streaming rate first increases sharply, since at this point the source uplink capacity bounds the streaming rate. Later the streaming rate (almost) flattens out, since now all peers' capacities have been used up and the extra rate for each user must come from the source directly. From such curves we can determine the most efficient source uplink capacity: the smallest value at which all peers' capacities are (almost) fully utilized.

B. Streaming Rate

We study the impact of the degree bound M on the maximum streaming rate, total node degree per node across all the trees, and maximum receiver delay. The results in the res to this section are for the single session scenario on a full-mesh graph with degree bound M on all nodes (including the source node) in each tree. They are obtained using the primal-dual algorithm in Section 3 with optimality guarantee of $\theta = 10\%$. The source node uplink capacity is fixed at 768 Kbps and the uplink capacities of other nodes are chosen according to the empirical distribution above.

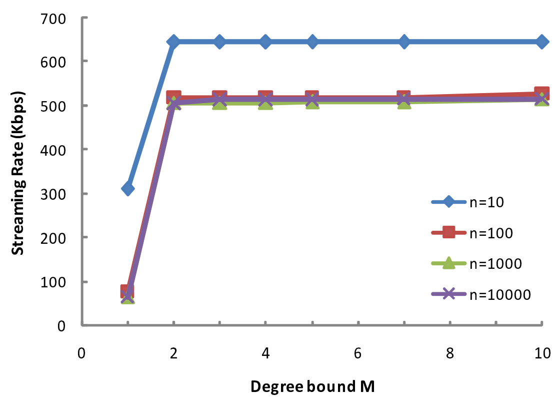

Fig. 10. Streaming Rate (Kbps) vs. Degree Bound M.

In Figure 10, we plot the maximum streaming rate as a function of the degree bound M for the different topologies. We observe a big jump in streaming rate as the degree bound is relaxed from $M = 1$ to $M = 2$. The streaming rate (approximately) flattens out at $M = 2$ onwards in this example. The rate for the $n = 10$ topology is higher than that of the others, simply due to the random sampling of uplink capacities for a *small number of nodes* resulting in a higher fraction of 768 Kbps uplink capacity nodes.

C. Node Total Degree

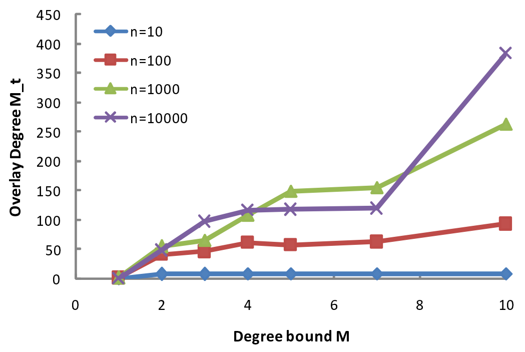

Fig. 11. Average Degree Usage in Overlay vs. Degree Bound M.

Since multiple trees support the same streaming session, the total number of children a node has is the sum of the number of children in all the trees. We refer to this as the total-out-degree of a node. In Figure 11, we plot the average node total-out-degree as a function of the degree bound M for the different topologies. At degree bound $M = 2$, the average degree usage in the overlay is about 50 for topologies with $n >= 100$ nodes. For $M = 5$ onwards, we see that the overlay degree can be up to an order of magnitude higher than the degree bound M per tree – about few hundreds for the larger networks ($n = 1000, 10000$). This suggests that for large topologies, bounding the out-degree per tree is not effective in keeping the total-out-degree in the overlay to a small value. Future work needs to address the challenging issue of computing P2P streaming capacity under total-out-degree across for each node across all the trees.

We also make the following observation about the effect of the approximation parameter θ in the primal-dual algorithm on the total-out-degree distribution. If we set θ to be very small (e.g., 1%), it leads to the usage of additional trees with very small rates assigned to them. This increases the streaming rate a little while raising the node total-out-degree significantly. Given the resource constraints such as CPU per node, it is desirable to avoid too large a total-out-degree, implying that θ should not be too close to zero.

D. Delay

In Figure 12, we plot the maximum receiver delay as a function of the degree bound M for the different topologies. Here, delay for a receiver is the maximum delay experienced by that receiver across all trees that it receives data from. For each tree, the delay is the time it takes from a packet originating at the source of the tree to its reaching the receiver. This delay is a function of both propagation delay, which is set at 20

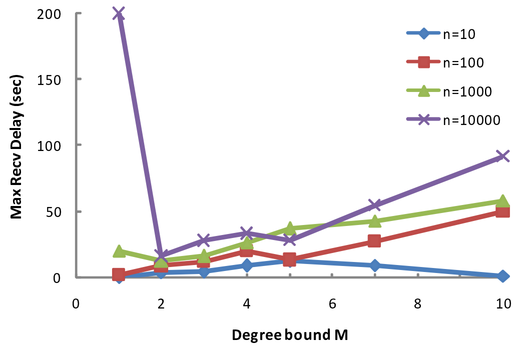

Fig. 12. Maximum Receiver Delay (sec) vs. Degree Bound M.

ms between two directly connected peers in this example, and fan-out delay at a node, which is the by-product of the peering relationships of the multicast trees. The fan-out delay at a node in a tree is the delay involved in pushing out a packet to each of its children peers sequentially. For example, for a packet of 1 kB, if this node uses an uplink rate of u bps to distribute content on this tree, then the beginning of data transfer to the i-th child node starts with a delay of $[(i-1)*1000*8/u]$ seconds – this isthe fan-out delay experienced by the i-th child node. Clearly, minimizing receiver delay involves a tradeoff between minimizing propagation delay through small tree depth and minimizing fan-out delay through low out-degree.

From Figure 12, we observe that for all networks considered, the delay is under 1 min and of the order of few tens of seconds, for degree bound values from $M = 2$ to $M = 5$. Beyond $M = 5$, the contribution of fan-out delay leads to a larger increase in maximum delay.

VII. Conclusion

P2P is becoming an essential part of streaming applications on the Internet, with applications ranging from IPTV to video conferencing. But just how much content can P2P systems stream to all the users remains unknown, in part because of the difficult combinatorial problems involved in multicast tree construction. This paper presents the first comprehensive framework to answer this question. We provide a taxonomy of sixteen problem formulations and develop a general framework of iterative primal-dual algorithms for computing the P2P streaming capacity (or, capacity region in the multiple sessions case), under topology constraints, node degree bounds, and possible presence of helpers. The results can be used to quantify the impact of these factors on P2P capacity in streaming applications. To complete each step in the primal-dual updates, we also develop efficient and accurate algorithms to compute smallest price trees, as guided by the given dual variables at that iteration and further guiding the primal-dual-variable update in the next iteration. For some of the formulations, these algorithms benchmark P2P streaming systems in polynomial time rather than the apparent need for exponential time search of multi-trees. For two formulations, polynomial time computation of P2P streaming capacity remains open.

Acknowledgments

We are thankful to Gary Chan, Wenjie Jiang, Xing Jin, David Johnson, Jennifer Rexford, and Rui Zhang-Shen for helpful discussions and collaborations.

Appendix

A. Correctness and complexity proof: Single session

We begin with some notation, then state some useful lemmas, and finally conclude with the proof of Theorem 1 from the previous section. The proof is adapted from techniques used in [6].

Given a set of dual weights $p(v)$, let $D(p)$ denote the dual objective function value and let $\Gamma(p)$ denote the minimum value of the LHS of dual program constraint (6) over all trees $t \in T$. Then, solving the dual program is equivalent to finding a set of weights $p(v)$ such that $D(p)/\Gamma(p)$ is minimized. Denote the optimal objective function value of the latter by θ, i.e., $\theta = \min_p D(p)/\Gamma(p)$.

We introduce some more notation before stating an important lemma. Let p_{i-1} denote the weight function at the beginning of iteration i of the **while** loop, and let A_{i-1} be the value of $\sum_{t\in T} y_t$ (primal objective function) up to the end of iteration $i-1$. Suppose the algorithm terminates after iteration L.

Lemma 6. *At the end of every iteration i, $1 \le i \le L$ of the Single-Session Primal-Dual Algorithm, the following holds*

$$D(p_i) \le N\delta \prod_{j=1}^{i} [1 + \frac{\varepsilon}{\theta}(A_j - A_{j-1})]$$

Proof: Let $t = \bar{t}$ be the node for which $V(t)$ is minimum. Recall that the weights are updated as

$$p_i(v) = p_{i-1}(v)(1 + \frac{\varepsilon m_{v,\bar{t}} y}{C(v)}) \ \forall\ v \in I(\bar{t})$$

where y is the total flow sent on tree t during iteration i. Using this, we have

$$\begin{aligned} D(p_i) &= \sum_{v\in V} C(v) p_i(v) \\ &= \sum_{v\in V} C(v) p_{i-1}(v) + \varepsilon \sum_{v\in V} p_{i-1}(v) m_{v,\bar{t}} y \\ &= D(p_{i-1}) + \varepsilon \sum_{t\in T} p_{i-1}(v) m_{v,\bar{t}} y \\ &= D(p_{i-1}) + \varepsilon y \Gamma(p_{i-1}) \\ &= D(p_{i-1}) + \varepsilon (A_i - A_{i-1}) \Gamma(p_{i-1}) \end{aligned}$$

Using this for each iteration down to the first one, we have

$$D(p_i) = D(p_0) + \varepsilon \sum_{j=1}^{i} (A_j - A_{j-1}) \Gamma(p_{i-1}) \quad (22)$$

From the definition of θ, we have $\theta \le \frac{D(p_{j-1})}{\Gamma(p_{j-1})}$, whence $\Gamma(p_{j-1}) \le \frac{1}{\theta} D(p_{j-1})$. Also, $D(p_0) = N\delta$. Using these in equation (22), we have

$$D(p_i) \le N\delta + \frac{\varepsilon}{\theta} \sum_{j=1}^{i} (A_j - A_{j-1}) D(p_{i-1}) \quad (23)$$

The property claimed in the lemma can now be proved using inequality (23) and mathematical induction on the iteration number i. We omit the details here, but point out that the induction basis case (iteration $i = 1$) holds since $p_0(v) = \frac{\delta}{C(v)}\ \forall\ v \in V$ and $D(p_0) = N\delta$. ■

We now estimate the factor by which the objective function value value A_L in the primal solution when the algorithm terminates needs to be scaled to ensure that link capacity constraints are not violated.

Lemma 7. *When the Single-Session Primal-Dual Algorithm terminates, the primal solution needs to be scaled by a factor of at most $\log_{1+\varepsilon} \frac{1+\varepsilon}{\delta}$ to ensure primal feasibility.*

Proof: Consider the uplink of any node v and associated weight $p(v)$. The value of $p(v)$ is updated when flow is augmented on uplink of node v. Let the sequence of flow augmentations (per iteration) on uplink of node v be $\Delta_1, \Delta_2, \ldots, \Delta_k$, where $k \le L$. Let $\sum_{i=1}^{k} \Delta_i = \kappa C(v)$, i.e., the total flow routed on uplink of node v exceeds its capacity by a factor of κ.

Because of the way in which augmented flow y is chosen in accordance with equation (9), we have $\Delta_i \le C(v)$ for all i. Hence, the dual weight $p(v)$ is updated by a factor of at most $1+\varepsilon$ after each iteration. Since the algorithm terminates when $D(p) \ge 1$, and since dual weights are updated by a factor of at most $1+\varepsilon$ after each iteration, we have $D(p_L) < 1+\varepsilon$. Since the weight $p(v)$, with coefficient $C(v)$, is one of the summing components of $D(p)$, we have $C(v)p_L(v) < 1+\varepsilon$. Also, the value of $p_L(v)$ is given by

$$p_L(v) = \delta \prod_{i=1}^{k} (1 + \frac{\Delta_t}{C(v)} \varepsilon)$$

Using the inequality $(1+cx) \ge (1+x)^c\ \forall\ x \ge 0$ and any $0 \le c \le 1$ and setting $x = \varepsilon$ and $c = \frac{\Delta_t}{C(v)} \le 1$, we have

$$\begin{aligned} \frac{1+\varepsilon}{C(v)} > p_L(v) &\ge \delta \prod_{i=1}^{k} (1+\varepsilon)^{\Delta_t / C(v)} \\ &= \delta (1+\varepsilon)^{\sum_{i=1}^{k} \Delta_t / C(v)} \\ &= \delta (1+\varepsilon)^{\kappa} \end{aligned}$$

whence,

$$\kappa < \log_{1+\varepsilon} \frac{1+\varepsilon}{\delta}$$

■

Proof of Theorem 1: Using Lemma 6 and the inequality $1+x \le e^x\ \forall\ x > 0$, we have

$$\begin{aligned} D(p_i) &\le N\delta \prod_{j=1}^{i} e^{\frac{\varepsilon}{\theta}(A_j - A_{j-1})} \\ &= N\delta e^{\varepsilon A_t / \theta} \end{aligned}$$

The simplification in the above step uses telescopic cancellation of the sum $(A_j - A_{j-1})$ over j. Since the algorithm terminates after iteration L, we must have $D(p_L) \ge 1$. Thus,

$$1 \le D(p_L) \le N\delta e^{\varepsilon A_i / \theta}$$

whence,

$$\frac{\theta}{A_L} \le \frac{\varepsilon}{\ln \frac{1}{N\delta}} \quad (24)$$

From Lemma 7, the objective function value of the feasible primal solution after scaling is at least

$$\frac{A_L}{\log_{1+\varepsilon}\frac{1+\varepsilon}{\delta}}$$

The approximation factor for the primal solution is at most the gap (ratio) between the primal and dual solution. Using (24), this is given by

$$\begin{aligned}\frac{\theta}{A_L} &\leq \frac{\varepsilon\log_{1+\varepsilon}\frac{1+\varepsilon}{\delta}}{\ln\frac{1}{N\delta}}\\ &= \frac{\varepsilon}{\ln(1+\varepsilon)}\frac{\ln\frac{1+\varepsilon}{\delta}}{\ln\frac{1}{N\delta}}\end{aligned}$$

The quantity $\ln\frac{1+\varepsilon}{\delta}/\ln\frac{1}{N\delta}$ equals $1/(1-\varepsilon)$ for $\delta = (1+\varepsilon)/[(1+\varepsilon)N]^{1/\varepsilon}$. Using this value of δ, the approximation factor is upper bounded by

$$\frac{\varepsilon}{\ln(1+\varepsilon)}\frac{1}{(1-\varepsilon)} \leq \frac{\varepsilon}{(\varepsilon-\varepsilon^2/2)(1-\varepsilon)} \leq \frac{1}{(1-\varepsilon)^2}$$

Setting $1+\zeta = 1/(1-\varepsilon)^2$ and solving for ε, we get the value of ε stated in the theorem.

To obtain the running time for the Single-Session Primal-Dual Algorithm, we first consider the running time of each iteration of the algorithm during which a tree $\bar{t}$ is chosen to augment flow. Selection of this tree involves a smallest price tree computation which takes T_{spt} time (say). All other operations within an iteration are absorbed (up to a constant factor) by the time taken for this smallest price computation, leading to a total of $O(T_{spt})$ time per iteration.

We next estimate the number of iterations before the algorithm terminates. Recall that in each iteration, flow y is augmented along the tree tv such that the total flow sent on any node uplink v during that iteration is at most $C(v)$. Thus, for at least one node v, $m_{v,t}y = C(v)$ and $p(v)$ increases by a factor of $1+\varepsilon$. Accordingly, with each iteration, we can associate a weight $p(v)$ which increases by a factor of $1+\varepsilon$.

Consider the weight $p(v)$ for fixed $v \in V$. Since $p_0(v) = \frac{\delta}{C(v)}$ and $p_L(v) \leq \frac{1+\varepsilon}{C(v)}$ (as deduced in the proof of Lemma 2), the maximum number of times that this weight can be associated with any iteration is

$$\log_{1+\varepsilon}\frac{1+\varepsilon}{\delta} = \frac{1}{\varepsilon}(1+\log_{1+\varepsilon}N) = O(\frac{1}{\varepsilon}\log_{1+\varepsilon}N)$$

Since there are a total of n weights $C(v)$, hence the total number of iterations is upper bounded by $O(\frac{N}{\varepsilon}\log_{1+\varepsilon}N)$. Multiplying this by the running time per iteration, we obtain the overall algorithm running time as $O\left(\frac{N}{\varepsilon^2}T_{spt}\log N\right)$. ■

B. Correctness and complexity proof: Multiple sessions

We first introduce some notation, then state some useful lemmas, and finally provide the proof of Theorem 2 from the previous section. The proof is adapted from techniques used in [6].

Recall that the Multi-Session Primal-Dual Algorithm runs in phases. In each phase, we route r^k units for session k from source s^k to receivers in R^k during iteration k for each $k = 1,2,\ldots,K$. The rate r^k for session k in phase i is routed in multiple steps. In step s of phase i of session k, we compute a smallest price tree $t \in T^k$ (under current value of prices p_v for all $v \in V$) and route flow along the associated tree. The weights $p(v)$ for nodes in the tree t are then adjusted as described earlier.

Let $r^k_{i,s}$ represent the remaining amount of flow to be sent in steps $s+1, s+2, \ldots$ in phase i for session k. Since we have to route r^k units of flow for session k in each phase, we have $r^k_{i,0} = r^k$ for all phases i. Let $p^{i,k,s-1}$ represent the node prices at the beginning of step s for routing session k during phase i. In this step, we determine the tree in T^k with smallest price

$$price_k(p^{i,k,s}) = \min_{t\in T^k}\sum_{v\in V} m_{v,t}p^{i,k,s}(v)$$

Denote this tree by $t^{i,k,s}$ and the amount of flow routed on it by $f^{i,k,s}$. Then, $r^k_{i,s} = r^k_{i,s-1} - f^{i,k,s}$. The prices associated with each node v are updated as

$$p^{i,k,s}(v) = p^{i,k,s-1}(v)\left(1+\frac{\varepsilon m_{v,t^{i,k,s}}f^{i,k,s}}{C(v)}\right)$$

To simplify the notation, we will drop the superscript (corresponding to step number) when we refer to prices at the end of routing one session during a phase or at the end of a complete phase, e.g., $p^{i,k}(v)$ will denote the price after routing session k during phase i, and $p^{i,K}(v)$ will denote the same after completion of phase i.

Let $D(p)$ denote the dual objective function value. The Multi-Session Primal-Dual Algorithm terminates at the first phase ρ for which $D(p^{\rho,K}) \geq 1$. Define a function Γ on the prices that computes the LHS of (16) of the dual program, that is

$$\Gamma(p) = \sum_{k=1}^{K} r^k price_k(p)$$

Then, solving the dual program is equivalent to finding a set of weights p_v such that $D(p)/\Gamma(p)$ is minimized. Denote the optimal objective function value for the latter by β, i.e., $\beta = \min_p D(p)/\Gamma(p)$.

Lemma 8. *When the Multi-Session Primal-Dual Algorithm terminates, the primal solution needs to be scaled by a factor of at most* $\log_{1+\varepsilon}(1/\delta)$ *to ensure primal feasibility (i.e., satisfying node uplink capacity constraints).*

Proof: Consider any node v and associated price $p(v)$. The value of $p(v)$ is updated when flow is augmented on the uplink of node v because of routing on a tree. Let the sequence of flow augmentations on the uplink of node v be $\Delta_1, \Delta_2, \ldots, \Delta_\ell$. Let $\sum_{i=1}^{\ell}\Delta_i = \kappa C(v)$, i.e, the total flow routed on the uplink of node v exceeds its capacity by a factor of κ.

Since the algorithm terminates when $D(p) \geq 1$, and since prices are updated by a factor of at most $1+\varepsilon$ after each iteration, we have $D(p^{\rho-1,K}) < 1$ and $D(p^{\rho,K}) < 1+\varepsilon$. Since the quantity $C(v)p(v)$ is one of the summing components of $D(p)$, hence $C(v)p^{\rho-1,K}(v) < 1$. Also, the value of $p^{\rho-1,K}(v)$ is given by

$$p^{\rho-1,K}(v) = \frac{\delta}{C(v)}\prod_{i=1}^{\ell}(1+\frac{\Delta_i}{C(v)}\varepsilon)$$

Using the fact that $(1+cx) \geq (1+x)^c\ \forall\, x \geq 0$ and any $0 \leq c \leq 1$ and setting $x = \varepsilon$ and $c = \frac{\Delta_i}{C(v)} \leq 1$, we have

$$\begin{aligned} 1 \;>\; C(v)p^{\rho-1,K}(v) &\geq \delta \prod_{i=1}^{\ell}(1+\varepsilon)^{\Delta_i/C(v)} \\ &= \delta(1+\varepsilon)^{\sum_{i=1}^{\ell}\Delta_i/C(v)} \\ &= \delta(1+\varepsilon)^{\kappa} \end{aligned}$$

whence,

$$\kappa < \log_{1+\varepsilon}\frac{1}{\delta}$$

■

Lemma 9. *At the end of ρ phases in the Multi-Session Primal-Dual Algorithm, we have*

$$\frac{\beta}{\rho-1} \leq \frac{\varepsilon}{(1-\varepsilon)\ln\frac{1-\varepsilon}{n\delta}}$$

Proof: We first derive inequalities to relate the values of $D(p)$ across consecutive steps during routing of a given session during a phase. Using this, the relation between values of $D(p)$ across consecutive phases is derived. At the end of step s for routing session k during phase i, we have

$$\begin{aligned} D(p^{i,k,s}) &= \sum_{v\in V} C(v)p^{i,k,s}(v) \\ &= \sum_{v\in V} C(v)p^{i,k,s-1}(v) + \varepsilon f^{i,k,s}\sum_{v\in V} m_{v,t^{i,k,s}}p^{i,k,s-1}(v) \\ &= D(p^{i,k,s-1}) + \varepsilon f^{i,k,s} price_k(p^{i,k,s-1}) \\ &\leq D(p^{i,k,s-1}) + \varepsilon f^{i,k,s} price_k(p^{i,k,s}) \end{aligned}$$

Note the use of the fact that the weights p_v are non-decreasing as the algorithm progresses. Summing the last inequality over all steps for routing session k during phase i, we have

$$D(p^{i,k}) \leq D(p^{i,k,0}) + \varepsilon r^k price_k(p^{i,k})$$

We now sum over all iterations during phase i to obtain

$$\begin{aligned} D(p^{i,K}) &\leq D(p^{i,0}) + \varepsilon\sum_{k=1}^{K} r^k price_k(p^{i,K}) \\ &= D(p^{i,0}) + \varepsilon\Gamma(p^{i,K}) \\ \text{or, } D(p^{i,K}) &\leq D(p^{i-1,K}) + \varepsilon\Gamma(p^{i,K}) \end{aligned}$$

Since $\beta \leq \frac{D(p^{i,K})}{\Gamma(p^{i,k})}$, we have

$$D(p^{i,K}) \leq \frac{D(p^{i-1,K})}{1-\frac{\varepsilon}{\beta}}$$

Using the initial value $D(p^{1,0}) = n\delta$, we have for $i \geq 1$

$$D(p^{i,K}) \leq \frac{n\delta}{1-\varepsilon}\, e^{\frac{\varepsilon(i-1)}{\beta(1-\varepsilon)}}$$

The last step uses the assumption that $\beta \geq 1$. The procedure stops at the first phase ρ for which

$$1 \leq D(p^{\rho,K}) \leq \frac{n\delta}{1-\varepsilon}\, e^{\frac{\varepsilon(\rho-1)}{\beta(1-\varepsilon)}}$$

which implies that

$$\frac{\beta}{\rho-1} \leq \frac{\varepsilon}{(1-\varepsilon)\ln\frac{1-\varepsilon}{n\delta}}$$

■

Proof of Theorem 3: Let γ represent the ratio of the dual to the primal solution. Then, we have

$$\gamma < \frac{\beta}{\rho-1}\log_{1+\varepsilon}\frac{1}{\delta}$$

Substituting the bound on $\frac{\beta}{\rho-1}$ from Lemma 9, we obtain

$$\gamma < \frac{\varepsilon\log_{1+\varepsilon}\frac{1}{\delta}}{(1-\varepsilon)\ln\frac{1-\varepsilon}{n\delta}} = \frac{\varepsilon}{(1-\varepsilon)\ln(1+\varepsilon)}\frac{\ln\frac{1}{\delta}}{\ln\frac{1-\varepsilon}{n\delta}}$$

Setting $\delta = \left(\frac{1-\varepsilon}{n}\right)^{\frac{1}{\varepsilon}}$, we get $\gamma \leq (1-\varepsilon)^{-3}$. Equating the desired approximation factor $(1+\zeta)$ to this ratio and solving for ε, we get the value of ε stated in the theorem.

To obtain the running time for the Multi-Session Primal-Dual Algorithm, we first upper bound the number of phases after which the algorithm terminates. Using weak-duality from linear programming theory, we have

$$1 \leq \gamma < \frac{\beta}{\rho-1}\log_{1+\varepsilon}\frac{1}{\delta}$$

Hence, the number of phases ρ is less than $1+\beta\log_{1+\varepsilon}(1/\delta)$, and is upper bounded by $\rho = \lceil\frac{\beta}{\varepsilon}\log_{1+\varepsilon}\frac{n}{1-\varepsilon}\rceil$.

For each step in an iteration, except possibly the last one, we increase the weight p_v associated with at least one node v by a factor of $1+\varepsilon$. Since each weight has an initial value of $\frac{\delta}{C(v)}$ and a final weight less than $\frac{1+\varepsilon}{C(v)}$, the number of steps exceeds the number of iterations by at most $n\log_{1+\varepsilon}\frac{1+\varepsilon}{\delta}$. Thus, the total number of steps is at most $(2K\log K + n)\lceil\frac{\beta}{\varepsilon}\log_{1+\varepsilon}\frac{n}{1-\varepsilon}\rceil$ and each such step takes T_{spt} time. Assuming that $\beta \leq 2$, the running time of the algorithm is obtained as stated in the theorem.

Using a technique similar to that outlined in [6], the node uplink capacities and/or session rates can be scaled (without incurring any additional running time overhead up to a constant factor) so that $1 \leq \beta \leq 2$. We omit the details here. Hence, the assumption on β is not restrictive.

□

C. Proof of SPT for full mesh graph without degree bound

We use p_s, p_r, and p_h to denote the price of the source s, the smallest price of receiver nodes, and the smallest price of the helper nodes. Clearly, SPT has no leaf helper nodes. If there is no helper node involved, then there are altogether $N+1$ nodes and N edges in the tree, and thus the total outgoing degree is N. As the root is the source, the source degree is at least 1. Denote the tree price by P_t, we have

$$P_t \geq p_s + (N-1)\min(p_s, p_r)$$

If there are m ($m \geq 1$) helper internal nodes in the tree, then there are altogether $N+m$ edges, and there are $N+m$ total outgoing degrees. We have

$$P_t \geq p_s + (N-1+m)\min(p_s,p_r,p_h) \geq\geq p_s + N\min(p_s,p_r,p_h)$$

So, we know that

$$P_t \geq p_s + \min((N-1)p_s, (N-1)p_r, Np_h),$$

Since the algorithm in Figure 4 reaches the right hand side of the above inequality, it is optimal.

D. Proof of SPT for full mesh graph with degree bound

Each tree has $N+1$ nodes and N edges (and thus N total outgoing degrees). As the root is the source, at least one edge is from the source. Index the first edge from the source by 1, and index the remaining $N-1$ edges from 2 to N. Denote the price of the l-th edge (the price of the outgoing node of the edge) by $p^{(l)}$, and denote the tree price by P_t, then

$$P_t = p_s + \sum_{l=2}^{N} p^{(l)}$$

Index all the $N+1$ nodes (no matter source or receivers) from 1 to $N+1$ by their prices, such that, $p_i \leq p_j, \forall i < j$. Then, for smallest price tree purpose, we should use as many low indexed nodes as possible for internal nodes, and we have

$$p_t \geq p_s + \sum_{i=1}^{I} p(i),$$

where I is the smallest number such that

$$\sum_{i=1}^{I} \tilde{M}(i) \geq N-1,$$

and $\tilde{M}(i) = M(i)$ if $i \in R$, and $\tilde{M}(i) = M(s)-1$ if i is the source. It is straightforward to see that the algorithm in Figure 5 constructs a tree whose tree price equals to the right hand side of the above inequality, so it is optimal.

E. Proof of SPT for general graph without degree bound

We start by proving the first claim. Let t be a directed Steiner tree in G rooted at s. We map t to a group Steiner tree t' in G' as follows:

- for source node s in t, we map it to s in G' and include s into t'.
- for every directed link in t that connects node v_1 to node v_2, we map it to node $n_{[v_1,v_2]}$ in G' and include $n_{[v_1,v_2]}$ into tree t'.
- for all the nodes $n_{[s,v]}$ in t', we include the link in G' that connects s and $n_{[s,v]}$ into t';
- for every intermediate node v in t with parent node v_p and a set of children, denoted by $Cld(v)$, we include the links connecting node $n_{[v_p,v]}$ and every $n_{[v,v_c]}$, $v_c \in Cld(v)$, into t' (Note these links in t' have the same price p_v).

Fig. 13 shows one example of the above mapping. Clearly the complexity is polynomial in the number of links in t.

We now show that t' is a tree. First, every link in t maps to a node in t', and the shared vertexes of any two links in t maps to the links connecting the corresponding nodes in t'. Hence t' is a connected subgraph.

Second, there is no loop in t'. Otherwise, there exist two nodes in t', between which there are two disjoint paths. This means there are two links in t between which there are two disjoint paths (i.e., a loop). But this is not possible since t is a tree and contains no loop.

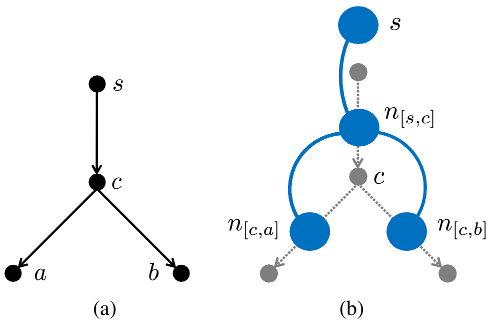

Fig. 13. a) A direct Steiner tree t in G connecting s, a and b; b) The group Steiner tree t' in G', mapped from t in a); for example, the edge from s to c in t maps to the node $n_{[s,c]}$ in t', and node c in t maps to two undirected edges connecting $n_{[s,c]}$ to $n_{[c,a]}$ and to $n_{[c,b]}$, respectively.

By this mapping, the source node s and every link in t maps to a node in t'. Consequently, for the node sets g_s and g_r, $r \in R$, in G', at least one node from each set (corresponding to at least one link coming out of s or going into r, $r \in R$) is included in t'. Therefore, t' is a group Steiner tree in G'.

Moreover, source node s and every intermediate node in t maps to a series of links in t', the number of which is exactly the number of children the node has in t. The price of t' is hence sum of the products of every node v's ($v \in t$) upload price and its number of children, and is the same as the price of t.

We now prove the second claim. Let t' be a group Steiner tree in G' connecting at least one node from each of node sets g_s and g_r, $r \in R$. We map it to a directed Steiner tree t in G as follows. This mapping is slightly complicated since now every node in t' corresponds to a pair of links in G and we need to specify which to map to.

- For source s in t', we map it to s in G and include s into t.
- we sort the nodes in t' into levels, according to their hop distances to s. That is, level i are the nodes i hops away from s.
- We map a node in level 1, say $n_{[s,v_1]}$, in t' to the directed link from s to its neighbor v_1 in G and include it into t. We then deal with $n_{[s,v_1]}$'s child nodes in the next level. If its name is of the form $n_{[v_1,v_2]}$, then we map it to the directed link from v_1 (s's neighbor) to v_2 in G and include it into t; otherwise, its name must be of the form $n_{[v_2,s]}$, and we map it to the directed link between s and v_2 in G and include it into t. We continue this process until every node in t' has been mapped into a directed link in t.
- for every directed link in t, we also include its two endpoint nodes into t. This way, all the nodes in G that are mapped from links in t' are included into T. The links in t are also connected since nodes are connected in t'. This way, t has the same cost as t', since all links in t' map to nodes in t and the number of children of a node in t is exactly the same as the number of links this node corresponds to in t'.
- we perform a breath first search to construct a directed

spanning tree in t rooted at s and reaching all nodes in t. We remove any leaf node in the resulted spanning tree that is not a receiver in R, as well as the directed links reaching these leaf nodes. We finally set t to be the pruned directed tree. This procedure can only reduce the price of t.

Fig. 14 shows one example of the above mapping. By above procedure, a group Steiner tree in T' maps to a directed Steiner tree in T with the same or less price. The complexity is again polynomial.

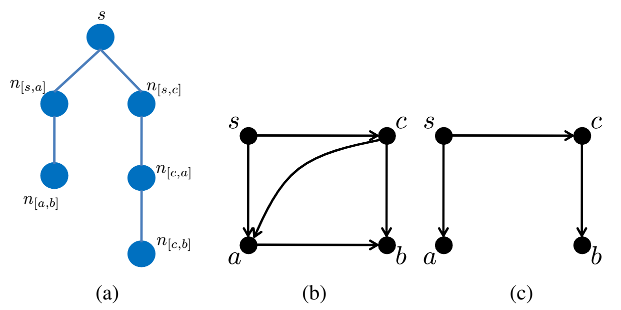

Fig. 14. a) A group Steiner tree t' in G' connecting at least one node from each of the groups: $\{s\}$, $\{n_{[s,a]}, n_{[a,b]}, n_{[c,a]}\}$ and $\{n_{[a,b]}, n_{[b,c]}\}$; b) The connected subgraph in G, mapped from t' in a); c) the final directed Steiner tree t in G after pruning the connected subgraph mapped from t' in a).

By the two claims, a minimum group Steiner tree in G', denoted by $\bar{t}$, can be mapped to a minimum directed Steiner tree in G, denoted by t^*, with the same price and vice versa, following the two mapping procedures we describe above.

We first prove the forward direction. Suppose it is not true. Following the second mapping procedure we describe above, $\bar{t}$ maps to a tree in G with a price higher than t^*. We then map t^* to a group Steiner tree in G' using the first mapping procedure. Then this new group Steiner tree has smaller price than $\bar{t}$, which contradicts with the setting that $\bar{t}$ has the minimum price. Hence, our assumption cannot be correct and $\bar{t}$ must map to a minimum cost Steiner tree in G.

The reverse direction can be proved by a similar set of arguments.

References

[1] D. Bertsimas and J. N. Tsitsiklis. *Introduction to Linear Optimization*. Athena Scientific, 1997.

[2] M. Charikar, C. Chekuri, T. Cheung, Z. Dai, A. Goel, S. Guha, and M. Li. Approximation algorithms for directed steiner tree problems. In *9th Annual ACM-SIAM SODA*, 1998.

[3] Y. J. Chu and T. H. Liu. On the shortest arborescence of a directed graph. *Science Sinica*, pages 1396–1400, 1965.

[4] Y. Cui, B. Li, and K. Nahrstedt. On achieving optimized capacity utilization in application overlay networks with multiple competing sessions. In *ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, June 2004.

[5] U. Feige. A threshold of ln n for approximating set cover. In *28th ACM STOC*, 1996.

[6] N. Garg and J. Konemann. Faster and simpler algorithms for multicommodity flow and other fractional packing problems. In *IEEE Symposium on Foundations of Computer Science*, 1998.

[7] N. Garg, G. Konjevod, and R. Ravi. A polylogarithmic approximation algorithm for the group steiner tree problem. In *ACM Symposium on Discrete Algorithms (SODA) '98*, 1998.

[8] C. Huang, J. Li, and K. W. Ross. Can internet video-on-demand be profitable? In *ACM SIGCOMM*, Kyoto, Japan, August 2007.

[9] R. Kumar, Y. Liu, and K. Ross. Stochastic fluid theory for p2p streaming systems. In *IEEE Infocom 2007*, April 2007.

[10] L. C. Lau, S. Naor, M. Salavatipour, and M. Singh. Survivable network design with degree or order constraints. In *39th ACM STOC*, 2007.

[11] J. Li, P. A. Chou, and C. Zhang. Mutualcast: an efficient mechanism for one-to-many content distribution. In *ACM SIGCOMM ASIA Workshop*, April 2005.

[12] S. Liu, M. Chiang, S. Sengupta, J. Li, and P. A. Chou. Performance bounds for large-scale heterogeneous p2p streaming system. *Allerton Conference 2008*, September 2008.

[13] S. Liu, R. Zhang-Shen, W. Jiang, J. Rexford, and M. Chiang. Performance bounds for peer-assisted live streaming. In *ACM SIGMETRICS 2008*, June 2008.

[14] T. Nguyen, K. Kolazhi, R. Kamath, S. Cheung, and D. Tran. Efficient multimedia distribution in source constraint networks. *IEEE Trans. on Multimedia*, 10(3):523–537, April 2008.