This full text paper was peer reviewed at the direction of IEEE Communications Society subject matter experts for publication in the IEEE INFOCOM 2008 proceedings.

Optimal Peer-to-Peer Technique for Massive Content Distribution

Xiaoying Zheng, Chunglae Cho and Ye Xia
Computer and Information Science and Engineering Department
University of Florida
Email: {xiazheng, ccho, yx1}@cise.ufl.edu

***Abstract*—A distinct trend has emerged that the Internet is used to transport data on a more and more massive scale. Capacity shortage in the backbone networks has become a genuine possibility, which will be more serious with fiber-based access. The problem addressed in this paper is how to conduct massive content distribution efficiently in the future network environment where the capacity limitation can equally be at the core or the edge. We propose a novel peer-to-peer technique as a main content transport mechanism to achieve efficient network resource utilization. The technique uses multiple trees for distributing different file pieces, which at the heart is a version of swarming. In this paper, we formulate an optimization problem for determining an optimal set of distribution trees as well as the rate of distribution on each tree under bandwidth limitation at arbitrary places in the network. The optimal solution can be found by a distributed algorithm. The results of the paper not only provide stand-alone solutions to the massive content distribution problem, but should also help the understanding of existing distribution techniques such as BitTorrent or FastReplica.**

***Index Terms*—Peer to Peer, Swarming, Content Distribution, Multicast, Optimization, Bandwidth allocation**

I. Introduction

A distinct trend has emerged that the Internet is used to transport data on a more and more massive scale. The first enabler for this trend is the on-going deployment of fiber-based access. For instance, Verizon will complete a nationwide fiber-to-the-home FTTH network in the US by 2010. The second enabler is the ever-increasing content size and quantity. Such massive content currently includes high-definition movies, live or recorded high-quality TV programs, large collection of multimedia data, and mountains of all automatically collected/sensed data such as environmental, scientific, or economic data. Beyond that, visionaries are already contemplating 3D super definition TV (643 Mbps) and 3D telepresence in virtual worlds.

With this trend, capacity shortage in the backbone networks has become a genuine possibility and may get increasingly more serious. As an evidence, with its early adoption of FTTH, by 2005, Japan already saw 62% of its backbone network traffic being from residential users to users, which was consumed by content downloading or P2P file sharing; the fiber users were responsible for 86% of the inbound traffic; and the traffic was rapidly increasing, by 45% that year [1]. The problem addressed in this paper is how to conduct massive content distribution efficiently in the future network environment where the capacity limitation can equally be at the core or the edge. In this view, efficient content distribution is not merely a service issue but a central networking issue.

We propose creating and applying novel P2P techniques as the main content transport mechanisms to achieve efficient network resource utilization. In particular, we will improve a class of techniques known as *swarming*. In a swarming session, the file to be distributed is broken into many chunks at the original source, which are then spread out across the peers. Subsequently, the peers can exchange the chunks with each other to speed up the distribution process. In this paper, swarming is viewed not just as a casual technique for end-user file-sharing applications. Instead, its benefits become clear if it is viewed as a fundamental networking/distribution mechanism, which can be applied to infrastructure-based content distribution. As will be seen, swarming can be thought as distributing content on *multiple* multicast trees. When done properly, it provides the most efficient utilization of the network capacity, a remedy for backbone congestion, or gives the fastest distribution.

For illustration of the main ideas in this paper, consider the toy example in Fig. 1. The numbers associated with the links are their capacities. Suppose a large file is split into many chunks at source node 1. We wish to find the fastest way to distribute all chunks to receivers 2 and 3. [1] We impose no restriction on how peers can help each other in the distribution process. Let us focus on a fixed chunk and consider how it can be distributed to the receivers. With some thoughts, it can be argued that, when the delay is not modeled, the path should be a tree rooted at the source and covering both receivers. All possible distribution trees are shown in Fig. 2. The question becomes how to assign the chunks to different distribution trees so that the distribution time is minimized, subject to the link capacity constraint. For this simple example, it is easy to see that distributing the chunks in 1:2 ratio on the second and third tree, while leaving the first unused, is optimal.

The ideas contained in the toy example are also given in [2]. That work focuses on how to compute the maximum throughput, which leads to the fastest distribution. Our contribution in this paper is on developing distributed algorithms to identify and use the optimal distribution trees, and at the same time, allocate correct bandwidth on the selected trees.

Our approach illustrated by the toy example can be contrasted with existing P2P distribution systems or techniques.

[1] As we will show, the objective of minimizing the distribution time is equivalent to maximizing the distribution throughput, which is also equivalent to minimizing the network congestion.

978-1-4244-2026-1/08/$25.00

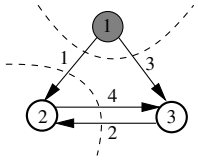


Fig. 1. Node 1 sends the file to node 2 and 3.

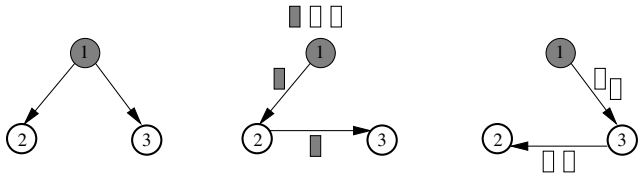


Fig. 2. All possible distribution trees for the example in Fig. 1.

A subset of examples include BitTorrent [3], FastReplica [4], Bullet [5], ChunkCast [6], and Julia [7]. Most existing systems are designed under the assumption that the access links are the bandwidth bottleneck. Hence, they may not deliver good performance under network bottleneck. Our solution will be able to automatically adapt to capacity constraint anywhere in the network. Furthermore, within our framework, we are able to solve four separate problems jointly and optimally: peer selection (from which peer to download), chunk selection (which chunks to download from a peer), distribution tree selection and bandwidth allocation. In contrast, other P2P systems solve one or two of these problems in isolation, typically using heuristic algorithms. Finally, the swarming technique is inherently more advanced and more powerful than all previous distribution schemes, including IP or application-level multicast (e.g. [8], [9]), network cache and existing content distribution networks (e.g., Akamai [10], CoBlitz [11]).

The paper is organized as follows. The models and problem formulations are given in Section II. The distributed algorithm is given in Section III. In Section III-D, we discuss practical issues in applying our algorithm to realistic settings, such as scalability and coping with network dynamics and churn. In Section IV, we evaluate the performance of our algorithm, including a comparison with BitTorrent and FastReplica. In Section V, we discuss additional related work. The conclusion is drawn in Section VI.

II. Problem Description

We will start with a formulation for optimal content distribution on a generic network. It turns out the problem is difficult on an arbitrary network. However, for overlay content distribution, the problem is far easier. We will give formulations for two possible scenarios of overlay distribution.

A. Optimal Multicast Tree Packing

Let the network be represented by $G = (V, E)$, where V is the set of nodes and E is the set of links. The capacity associated with each link $e \in E$ is c_e. The utilization of link e, a measure of link congestion, is denoted by μ_e. We define a multicast session as a group of nodes (members) exchanging the same file. In a session, some members own some distinct chunks of a large file (The case of overlapping content at different nodes requires a minor extension, which will be discussed in Section III-D1.) and we call those members *sources* of the session. A reasonable assumption about a session is that all members in the session are interested in the file, and at the end of file distribution, every member in the session will have a complete copy of the file.

Let M be the set of all multicast sessions. For each session $m \in M$, let $V^{(m)} \subseteq V$ represent the set of members in session m, and let $S^{(m)} \subseteq V^{(m)}$ be the set of sources in session m. For each source $s \in S^{(m)}$, let $L_s^{(m)}$ be the total size of the file chunks at source s for session m. Let the set of all possible multicast trees spanning all members in the session rooted at source $s \in S^{(m)}$ be denoted by $T_s^{(m)}$. A multicast tree may contain nodes not in the session, in which case the tree is called a *Steiner tree*. In the case where all nodes on the tree belong to the session, the multicast tree is called a *spanning tree*, meaning it spans the multicast session (rather than the whole network V). For the i^{th} tree $t_{s,i}^{(m)} \in T_s^{(m)}$, where the order of indexing is arbitrary, denote $z_{s,i}^{(m)}$ to be the sending rate on tree $t_{s,i}^{(m)}$ from the root s.

A straightforward objective is to minimize the overall downloading time for all multicast sessions, which is to minimize the worst downloading time associated with any source s in any session m. With some thought, we see that no difference is made in terms of the achievable downloading time if we assume that all sources in all sessions finish their distribution at the same time. We can then minimize this common duration t. Let the total rate on a link $e \in E$ be denoted by x_e. It is equal to the sum of all the rates on all the trees passing through link e, across all sessions and all sources. That is,

$$x_e = \sum_{m \in M} \sum_{s \in S^{(m)}} \sum_{i: e \in t_{s,i}^{(m)}} z_{s,i}^{(m)}. \tag{1}$$

The optimization problem is as follows.

$$\min \quad t \tag{2}$$

$$\text{s.t.} \quad t \sum_{i=1}^{|T_s^{(m)}|} z_{s,i}^{(m)} = L_s^{(m)}, \quad \forall s \in S^{(m)}, \ \forall m \in M \tag{3}$$

$$x_e \le c_e, \quad \forall e \in E \tag{4}$$

$$z_{s,i}^{(m)} \ge 0, \quad \forall i = 1, \cdots, |T_s^{(m)}|, \forall s \in S^{(m)}, \forall m \in M. \tag{5}$$

Condition (3) says that, if one looks at all the multicast trees rooted at a source s for a session m, the sum of the distribution rates on all these trees, multiplied by the distribution time, should be equal to the total size of all the file chunks stored at source s for session m. This means that every bit of the file stored at s must be transmitted exactly once. A moment of thinking reveals that nothing is gained by transmitting the same bit more than once. Condition (4) is the link capacity constraint. At each link e, the flow rate on the link should be no greater than the link capacity, c_e.

It turns out the above problem is equivalent to a minimizing-congestion problem. This is immediate if we define $y_{s,i}^{(m)} =$

$tz_{s,i}^{(m)}$ and make the substitution of variables. But, we will do this a little differently for ease of interpretation. Let $z_s^{(m)} = \sum_{i=1}^{|T_s^{(m)}|} z_{s,i}^{(m)}$ be the total sending rate at a source node s of session m. Select a set of constants $\{r_s^{(m)}\}$, each being proportional to $L_s^{(m)}$ with the same constant proportional factor. Each $r_s^{(m)}$ is understood as a rate. Consider a feasible solution $\{z_{s,i}^{(m)}\}$ and t. By (3), $z_s^{(m)}$ is proportional to $L_s^{(m)}$, the total size of the chunks at s for session m. Then, $z_s^{(m)} = \gamma r_s^{(m)}$, for some constant $\gamma > 0$. Next, define $\mu = 1/\gamma$. We then make the substitution of variables by $t = \frac{\mu L_s^{(m)}}{r_s^{(m)}}$, and redefine $z_{s,i}^{(m)}$ to be $\mu z_{s,i}^{(m)}$. We get the following minimizing-congestion formulation.

$$\min \quad \mu \tag{6}$$

$$\text{s.t.} \quad \sum_{i=1}^{|T_s^{(m)}|} z_{s,i}^{(m)} = r_s^{(m)}, \quad \forall s \in S^{(m)}, \ \forall m \in M \tag{7}$$

$$x_e \le \mu c_e, \quad \forall e \in E \tag{8}$$

$$z_{s,i}^{(m)} \ge 0, \quad \forall i = 1, \cdots, |T_s^{(m)}|, \forall s \in S^{(m)}, \forall m \in M. \tag{9}$$

In the above formulation, $r_s^{(m)}$ can be understood as the demanded rate, and μ is the maximum link utilization, which also measures the worst link congestion. The problem is to minimize the worst link congestion subject to the fulfillment of all demanded rates.

Thus, we have two equivalent views of optimal multicast tree packing. In the first view, the objective is to minimize the overall distribution time (or maximize the distribution throughput) while satisfying the link capacity constraint. In the second view, the objective is to best balance the network load while satisfying the rate demand for all sources and all sessions.

Another minor reformulation will be helpful later. Let μ_e stand for the utilization of link e, and let $\vec{\mu}$ denote the vector of μ_e over all links. Let $||\vec{\mu}||_\infty$ denote the maximum norm, i.e., $||\vec{\mu}||_\infty = \max_{e \in E} \mu_e$. The above minimizing-congestion formulation is equivalent to the following.

$$\min \quad ||\vec{\mu}||_\infty \tag{10}$$

$$\text{s.t.} \quad \sum_{i=1}^{|T_s^{(m)}|} z_{s,i}^{(m)} = r_s^{(m)}, \quad \forall s \in S^{(m)}, \ \forall m \in M \tag{11}$$

$$x_e = \mu_e c_e, \quad \forall e \in E \tag{12}$$

$$z_{s,i}^{(m)} \ge 0, \quad \forall i = 1, \cdots, |T_s^{(m)}|, \forall s \in S^{(m)}, \forall m \in M. \tag{13}$$

The optimization problem proposed so far is equivalent to the problem of packing Steiner trees [12], [13], which is computationally intractable. Fortunately, for P2P content distribution, the problem becomes simpler. The main reason is that the overlay network of each session consists of exactly those nodes in the session. Given such an overlay network, any Steiner tree that covers all nodes of the session is in fact a spanning tree. We will show later that the algorithm used to solve the optimization problem involves a minimum-cost spanning tree problem in each iteration. Should some Steiner node exist, it would have involved a minimum-cost Steiner tree problem. The former is far more tractable than the latter NP-hard problem. One should be reminded that, although the computation for the overlay network case is far easier, the achievable performance is sub-optimal since the overlay edges are determined by the fixed underlay routing.

Unlike the inflexible underlay routing, the bandwidth of the overlay edge may have alternatives. Next, we consider two cases, both of which can be useful.

B. Fixed Overlay Link Bandwidth

In this case, the bandwidth of each overlay link is a fixed constant[2] determined by a bandwidth allocation scheme external to our problem. For instance, the bandwidth may be determined by the end-to-end TCP control, or by bandwidth allocation algorithms that enforce other allocation policies such as the max-min fairness [14]. We assume the overlay nodes know about the bandwidth on each overlay link. For instance, in the case of TCP, the overlay link bandwidth can be measured. When the overlay link bandwidth is fixed, different distribution sessions become decoupled. We then have $|M|$ totally independent overlay networks. The original optimization problem becomes separated to $|M|$ identical but independent optimization problems. The solution to this problem will require running algorithms only at the overlay nodes, making deployment easy.

We illustrate this by focusing on one of the overlay networks, which corresponds to one session. Let $\hat{G} = (\hat{V}, \hat{E})$ represent the overlay network. For all other notations, since there is no danger of confusing them with earlier definitions, we will re-define them. The bandwidth associated with each overlay link $e \in \hat{E}$ is c_e, which is allocated already and is a constant. The utilization of overlay link e is denoted by μ_e. Assume $S \subseteq \hat{V}$ is the set of sources. Let T_s represent the set of all possible (overlay) multicast trees rooted at source s spanning all overlay nodes. Let $t_{s,i} \in T_s$ be the i^{th} (overlay) multicast tree and $z_{s,i}$ be the associated sending rate on tree $t_{s,i}$. The rate on the overlay link $e \in \hat{E}$ is

$$x_e = \sum_{s \in S} \sum_{i: e \in t_{s,i}} z_{s,i} \tag{14}$$

The optimization problem is now

$$\begin{aligned} &\min ||\vec{\mu}||_\infty \\ &\text{s.t.} \sum_{i=1}^{|T_s|} z_{s,i} = r_s, \quad \forall s \in S \\ &x_e = \mu_e c_e, \quad \forall e \in \hat{E} \\ &z_{s,i} \ge 0, \quad \forall i = 1, \cdots, |T_s|, \quad \forall s \in S. \end{aligned} \tag{15}$$

C. Optimally Allocated Overlay Bandwidth

In this subsection, we consider an alternative scenario with better performance. Instead of relying on TCP to allocate the overlay bandwidth, leading to the partition of the overall

[2]By fixed overlay bandwidth, we do not mean the bandwidth may not vary over time. We simply mean that the overlay link bandwidth is not determined by our algorithm and is known at the time of running the algorithm.

network into multiple independent overlay networks, we will incorporate overlay bandwidth allocation into the optimization problem. Note that different sessions are coupled together by the sharing of the underlay links. The solution to this problem will require cooperation from the physical links. But, it is possible to modify the problem slightly and run the algorithm at only bottleneck links, such as the inter-ISP slow links.

Let $\hat{G}^{(m)} = (\hat{V}^{(m)}, \hat{E}^{(m)})$ represent the *overlay network* for each session m. For each overlay link $\hat{e} \in \hat{E}^{(m)}$, the notation $e \in \hat{e}$ for some underlay link $e \in E$ means that link e is on overlay link $\hat{e}$, which is itself an underlay path. For each session m, let $T_s^{(m)}$ be the set of all possible spanning trees on $\hat{G}^{(m)}$ rooted at source s , and $t_{s,i}^{(m)}$ be the i^{th} tree in $T_s^{(m)}$. Then, the total rate on a physical link $e \in E$, x_e, is given by

$$x_e = \sum_{m \in M} \sum_{s \in S^{(m)}} \sum_{\hat{e} \in \hat{E}^{(m)}: e \in \hat{e}} \sum_{i: \hat{e} \in t_{s,i}^{(m)}} z_{s,i}^{(m)}. \qquad (16)$$

The optimization problem is exactly written as in (10)-(13).

III. Distributed Algorithm: Diagonally Scaled Gradient Projection

A. Fixed Overlay Link Bandwidth

The goal of minimizing $||\vec{\mu}||_\infty$ is to balance the network load. The same objective can be achieved by minimizing $\sum_{e \in \hat{E}} \hat{f}_e(x_e)$, where $\hat{f}_e$ is some convex increasing function on $x_e \geq 0$. Such an objective function discourages large link rate. One such function is the q norm $||\vec{\mu}||_q = (\sum_{e \in \hat{E}} \mu_e^q)^{1/q}$. In fact, $||\vec{\mu}||_\infty$ can be approximated by $||\vec{\mu}||_q$: as $q \to \infty$, $||\vec{\mu}||_q \to ||\vec{\mu}||_\infty$. We will assume $q \geq 2$ throughout. Since $\min ||\vec{\mu}||_q$ is equivalent to $\min ||\vec{\mu}||_q^q$, after substitution of μ_e with $\frac{x_e}{c_e}$, (15) becomes

$$\min \sum_{e \in \hat{E}} (\frac{x_e}{c_e})^q \qquad (17)$$

$$\text{s.t.} \sum_{i=1}^{|T_s|} z_{s,i} = r_s, \quad \forall s \in S \qquad (18)$$

$$z_{s,i} \geq 0, \quad \forall i = 1, \cdots, |T_s|, \quad \forall s \in S. \qquad (19)$$

For optimization problems with the simplex constraint of the form (18), the optimality condition is especially simple [15]. It has been shown in [16] and [17] that there exists a special gradient projection algorithm. For our case, the gradient projection algorithm can also be easily extended to an equally simple scaled version. The latter overcomes the issue that our problem may be ill-conditioned, and hence, drastically improves the algorithm's convergence time. Our computational experiences have shown that the scaled gradient algorithm is much faster than the unscaled algorithm or the subgradient algorithm. The latter is often used in network optimization problems.

Another difficulty is the large number of possible spanning trees, and hence, the large number of variables. Fortunately, the algorithm does not maintain all possible spanning trees. The following steps take place for every source at the overlay network level. The algorithm starts out with one or few spanning trees. In each iteration, a cost is assigned to each (overlay) link to reflect the current link congestion. Then, a minimum-cost spanning tree can be computed. The algorithm shifts an appropriate amount of traffic (rate) from each currently maintained spanning tree to the minimum-cost tree. The new minimum-cost tree enters the current collection of spanning trees. Some previous spanning tree may leave the collection if its distribution rate is reduced to zero.

We next illustrate some details. Let $T = \bigcup_{s \in S} T_s$ be the collection of all multicast trees rooted at any source. Let z be the vector $(z_{s,i})$ where $s \in S, i = 1, \ldots, |T_s|$, with an arbitrary indexing order for the sources. In problem (17), let the feasible set defined by (18) and (19) be denoted by $\mathcal{Z}$.

For each overlay link $e \in \hat{E}$, recall that x_e is the aggregate flow rate it carries. Let x be the vector $(x_e)_{e \in \hat{E}}$. Let H denote the $|\hat{E}| \times |T|$ link-tree incidence matrix associated with the trees in T (i.e. $[H]_{et} = 1$ if link e lies on tree t; and $[H]_{et} = 0$ otherwise). Obviously, $x = Hz$. Now define $\hat{f}_e(x_e) = (x_e/c_e)^q$ and $\hat{f}(x) = \sum_{e \in \hat{E}} \hat{f}_e(x_e)$. The objective function, denoted by $f(z)$, is given by

$$f(z) = \hat{f}(Hz) = \sum_{e \in \hat{E}} \hat{f}_e(x_e) = \sum_{e \in \hat{E}} (\frac{x_e}{c_e})^q, \qquad (20)$$

and (17) can be written as

$$\begin{aligned} \min \quad & f(z) = \hat{f}(Hz) \\ \text{s.t.} \quad & z \in \mathcal{Z}. \end{aligned} \qquad (21)$$

The derivative of the objective function $f(z)$ with respect to $z_{s,i}$ is given by

$$\frac{\partial f(z)}{\partial z_{s,i}} = \sum_{e \in t_{s,i}} \frac{\partial \hat{f}(x_e)}{\partial x_e} = \sum_{e \in t_{s,i}} q c_e^{-q} x_e^{q-1}. \qquad (22)$$

Note that $\frac{\partial \hat{f}(x_e)}{\partial x_e}$ is the so-called first-derivative link cost of link e [15], [17]. It reflects the current congestion level at link e. $\frac{\partial f(z)}{\partial z_{s,i}}$ is the first-derivative cost of the tree $t_{s,i}$, which is equal to the sum of the first-derivative costs of the links on the tree. It reflects the current congestion level of the tree $t_{s,i}$. The first-derivative tree cost is an important quantity. We will see later that our algorithm is to shift flows from trees with higher costs to the minimum-cost tree.

For each $s \in S$, let i_s be the index of a minimum-cost tree rooted at s (i.e., with s as the source). That is,

$$i_s(z) = \text{argmin}_{\{i: t_{s,i} \in T_s\}} \{\frac{\partial f(z)}{\partial z_{s,i}}\}. \qquad (23)$$

If there are multiple minimum-cost trees, we choose an arbitrary one. Since the feasible set $\mathcal{Z}$ is a convex set and the objective function is a convex function, we can characterize an optimal solution z^* to the problem (17) by the following optimality condition.

$$\sum_{s \in S} \sum_{i: t_{s,i} \in T_s} \frac{\partial f(z^*)}{\partial z_{s,i}} (z_{s,i} - z_{s,i}^*) \geq 0, \ \forall z \in \mathcal{Z}. \qquad (24)$$

This optimality condition might be equivalently written as, for any source $s \in S$,

$$z_{s,i}^* > 0 \text{ only if } [\frac{\partial f(z^*)}{\partial z_{s,j}} \geq \frac{\partial f(z^*)}{\partial z_{s,i}}, \forall t_{s,j} \in T_s]. \qquad (25)$$

That is, for every source, only those trees with the minimum first-derivative cost carry positive amount of flow. This intuitively suggests that, in the algorithm, we should shift flow to the minimum-cost trees from other trees.

It turns out this is exactly what the gradient projection algorithm does. We develop the gradient projection algorithm following the procedure in [17] to solve the problem (17). But we add diagonal scaling to speed up the algorithm's convergence time. Let $i_s(k)$ be a short hand for $i_s(z(k))$.

Diagonally Scaled Gradient Projection Algorithm

$$z(k+1) = \alpha(k)\bar{z}(k) + (1-\alpha(k))z(k) \tag{26}$$

$$\bar{z}_{s,i}(k) = \begin{cases} [z_{s,i}(k) - \delta(k)\cdot(d_{s,i}(k))^{-1}\cdot(\frac{\partial f(z(k))}{\partial z_{s,i}} - \frac{\partial f(z(k))}{\partial z_{s,i_s(k)}})]_+, \\ \qquad \text{if } i \neq i_s(k) \text{ and } z_{s,i}(k) \geq 0; \\ r_s - \sum_{1\leq j\leq |T_s|, j\neq i_s(k)} \bar{z}_{s,j}(k), \quad \text{if } i = i_s(k), \end{cases} \tag{27}$$

with

$$d_{s,i}(k) = \sum_{e\in t_{s,i}\cup t_{s,i_s(k)}\setminus t_{s,i}\cap t_{s,i_s(k)}} q(q-1)c_e^{-q}(x_e(k))^{q-2}. \tag{28}$$

In (26), $\alpha(k)$ is a scalar on $[\underline{a}, 1]$, for some $\underline{a}$, $0 < \underline{a} \leq 1$. (26) says that the new rate vector at the $(i+1)^{th}$ iteration, $z(k+1)$, is on the line segment between $z(k)$ and $\bar{z}(k)$.

The main part of the algorithm is expression (27), which computes the end point of a feasible direction, $\bar{z}(k)$, entry by entry. There are three cases.

- case 1 If a tree $t_{s,i}$ is not the chosen minimum-cost tree (with index $i_s(k)$) and $t_{s,i}$ has a positive flow, its rate will be reduced.
- case 2 If a tree $t_{s,i}$ is not the chosen minimum-cost tree and the tree has zero flow rate, then the rate stays at 0.
- case 3 If $t_{s,i}$ is the chosen minimum-cost tree, the rate of the tree is increased so that the total rates of all trees rooted at s will be equal to the demanded rate r_s.

Note that the description in case 3 ensures that $\bar{z}(k)$ is feasible (in $\mathcal{Z}$). Since $z(k)$ is also feasible, by (26), the new rate vector $z(k+1)$ is feasible. Hence, if we start with a feasible solution in $\mathcal{Z}$, $z(k)$ is in $\mathcal{Z}$ for all k.

What remains to be said is how much the rate is reduced in case 1. Note that the expression $\frac{\partial f(z(k))}{\partial z_{s,i}} - \frac{\partial f(z(k))}{\partial z_{s,i_s(k)}}$ is the difference in the first-derivative cost between the tree $t_{s,i}$ and the chosen minimum-cost tree, and the difference is always non-negative. Intuitively, the amount of reduction should be proportional to this difference. Indeed, if we ignore the factor $(d_{s,i}(k))^{-1}$ in (27), the rate reduction is proportional to the cost difference with the proportional constant (step size) $\delta(k) > 0$.

The factor $(d_{s,i}(k))^{-1}$ does the diagonal scaling, which is effective in dealing with our ill-conditioned problem, making the convergence much faster. It happens that the minimizing-congestion type of network problems is often ill-conditioned. In our case, the problem becomes more ill-conditioned when the parameter q in the q norm becomes larger. The scaling factor $(d_{s,i}(k))^{-1}$ can be understood as allowing different components of the vector z to use different step sizes in the iteration. The resulting scaled algorithm is far superior to the plain gradient projection algorithm.

The algorithm in (26)-(28) is a distributed one. In order to compute the tree cost, $\frac{\partial f(z)}{\partial z_{s,i}}$ in (22), and the scaling factor, $(d_{s,i}(k))^{-1}$ in (28), each link e can independently compute its corresponding term based on the local aggregate rate, x_e, passing through the link. Then, the tree cost and the scaling factor can be accumulated by the source s based on the link values along the tree. To find the minimum-cost tree $i_s(k)$, each source needs to compute the minimum-cost directed spanning tree (MDSP). Both centralized and distributed algorithms exist for computing the MDSP [18] [19] [20]. Both achieve $O(n^2)$ time complexity for a complete graph with n nodes. In the distributed version, the amount of information exchanged is also $O(n^2)$. In our implementation, each source collects the (overlay) link costs from all the receivers and uses a centralized algorithm to compute the MDSP. Other than that, the gradient algorithm is completely decentralized.

In addition to fast convergence, another strength of this gradient algorithm lies in that it avoids the enumeration of all possible spanning trees. The source only needs to manage the set of active multicast trees, i.e., those trees with positive flows. At each iteration, the source computes a new minimum-cost tree. A non-active tree won't become active unless it is the minimum-cost tree. The source only adjusts the flow rates of the active trees. The set of active trees usually is not very large if the algorithm converges fast, since, at each iteration, at most one more tree becomes active. For the original linear model (15), there are at most $|\hat{E}| + |S|$ active trees in any extreme point solution. But since this gradient algorithm is a kind of interior point method, strictly speaking, $|\hat{E}| + |S|$ is not really an upper bound. Nevertheless, it should give a rough sense on what the bound might be.

B. Optimally Allocated Overlay Bandwidth

The problem described in Section II-C can be worked out in a similar way, leading to a scaled gradient projection algorithm. Details vary a little and the notation is heavier. However, the resulting algorithm is still fully distributed and is quite simple. For brevity, we do not list the algorithm in this paper.

C. Convergence Results

We obtained theoretical conclusions that the scaled gradient algorithm converges and converges fast under the constant step size rule. These results, although conceptually simple, require technicality in order to be stated and proved correctly. For brevity, we omit some technical details in the statement of the results. We will also omit the proofs, which are an adaptation of the proofs in [21]. Moreover, we will only focus on the case of fixed overlay bandwidth. The same convergence results can be said for the case of optimally allocated bandwidth. In the theorems, $\delta_1 > 0$ is a known constant.

Therorem 1: (Global Convergence) For $0 < \delta < \delta_1$, every limit point of $\{z(k)\}$ generated by the synchronous gradient projection algorithm (26)-(28) is stationary.

We need an assumption for the next theorem.

- $A1$: At every optimum, the utilization of every overlay link is strictly positive.

Therorem 2: (Locally Linear Convergence Rate) Suppose $A1$ holds. For $0 < \delta \leq \delta_1$, there exist a closed ball $\hat{\mathcal{Z}}$ around an optimal solution z^*, and some $k_0 \geq 0$ where $z(k_0) \in \hat{\mathcal{Z}}$, the sequence $\{z(k)\}$ with $k \geq k_0$, generated by the synchronous gradient projection algorithm (26)-(28) converges to an optimal solution in the ball $\hat{\mathcal{Z}}$ and the convergence rate is linear (i.e., geometric).

The original optimization problem (15) can be approximated slightly differently. Note that $\min ||\vec{\mu}||_\infty$ has the same solution as $\min ||\vec{\mu} + \kappa||_\infty$, for any constant $\kappa > 0$. We then consider the approximation, $\min ||\vec{\mu} + \kappa||_q^q$. More precisely, we replace the objective function in (17) by $\min \sum_{e \in \hat{E}} (\frac{x_e + \kappa}{c_e})^q$ and keep the same constraints. We can then adapt the proof in [21] and conclude *global* geometric convergence.

D. Practical Considerations

The problem formulations in the previous sections omit some details that are practically required. The purpose of this omission is for ease of presentation. These simplified formulations contain the technical core, or the most difficult aspect, of the problem. For the most part, the formulations are without loss of generality. Practical details can be easily incorporated into the formulations. We now address several of these.

1) Overlapping Content: When some sources share overlapping chunks, we can create a virtual source node that connects all those sources and move all the overlapping chunks to the virtual source. The virtual source has one outgoing virtual link with infinite capacity connecting to each original source. We then arrive at an expanded network. Of course, in actual operation, one of the original physical sources will "act" as the virtual source and run the algorithms assigned to the virtual source.

2) Network Dynamics and Churn: Thus far, we assume that the network is stable and no members depart or join until all existing members finish downloading. Since we are considering the distribution of massive files or large collections of files, the assumption is reasonable for the most part. However, we do need to deal with low-degree member churn and network dynamics.

If any source owning unique chunks leaves before it finishes disseminating them, those chunks are no longer available in the network. To minimize such risk, in the optimization formulation, we can adjust the requested sending rates r_s of the sources. If any source is expected to leave the network soon, it may request (or be assigned) a higher sending rate so that it can spread its chunks to network more quickly.

Other types of network and member dynamics include link failures, the change of link capacities, the arrival of new sources, and the departure and arrival of receivers. As argued in [22], a distributed algorithm has built-in ability to adapt to variations. A distributed algorithm can react rapidly to a local disturbance at the point of disturbance with slower fine tuning in the rest of the network. Such adaptive ability is intimately connected with the algorithm's speed of convergence in the static case. Since our algorithm is the result of conscious effort to improve the convergence speed (by diagonal scaling of the gradient algorithm), we believe it is superior in coping with network and member dynamics compared to other similar distributed algorithms. In addition, our distributed algorithm is naturally robust because of the lack of reliance on a central node that might fail. In Section IV, we will show a small example of how our algorithm successfully adapts to the departure and arrival of receivers.

3) Scalability and Hierarchical Partition of Sessions: In each overlay network, the sources know the complete information of all overlay links and need to run the expensive centralized MDSP algorithm (There is a distributed MDSP algorithm [20]. But the price to pay is the potentially slower speed due to distributed operation.). Thus, our gradient algorithm can only deal with distribution sessions with limited size, say several thousands of members in each session. In order to improve the scalability of our algorithm, we shall partition each session and run the algorithm hierarchically, as most scalable network algorithms would do.

IV. Performance Evaluation

In this section we will compare the performance of our gradient algorithm (GP) with known theoretical bounds, BitTorrent (BT) and Adaptive FastReplica (AFR) [4]. We select BitTorrent because its techniques are interesting and it is the most popular P2P application. We select AFR because it can be thought as using multiple multicast trees for distribution. But only a subset of the trees are allowed, which we call two-level two-phase trees. In each of these trees, the source is connected to one receiver at level 1, and then the level 1 receiver is connected to all other receivers at level 2. It might appear that such a collection of trees is quite enough for achieving near optimal performance. In an access-constrained network, this is indeed true. However, we will show this is not the case for networks with interior bottlenecks. In that case, a different, maybe larger, collection of trees is needed.

In the previous sections, our objective function is the worst network utilization $||\vec{\mu}||_\infty$. In the evaluation part, we will focus on the source throughput $R_s = r_s/||\vec{\mu}||_\infty$, where r_s is the scaled sending rate, and the downloading time $t = L_s/R_s$, since these are what BitTorrent experiments yield directly. However, recall that the two measures are the two sides of the same coin.

The commercial ISP backbone and cross-ISP topologies are obtained from the Rocketfuel project [23].

a) BitTorrent: In the terminology of BitTorrent, a seed is a source, and a leecher is a receiver. We use the Bittorrent simulator developed by Bharambe et. al. [24]. Since the original simulator only supports access link constraint, we modified the simulator so that it supports general physical network topologies. The overlay link bandwidth, which is the per-connection bandwidth at the underlay, is determined by the max-min bandwidth allocation [14].

b) Adaptive FastReplica: AFR supports single source. To compare with AFR, we partition the physical network into several overlay networks, each for one source according to

max-min fairness allocation. AFR constructs two-phase trees as described earlier. From [4], the theoretical throughput of AFR in its best behavior can be computed.

c) Theoretical Optimum and Bounds: Some theoretical results are used as performance benchmark in our performance comparison. In several studies [25], [26], [27], researchers have analyzed a model of P2P file sharing among residential users in low access-speed environment. The source is to distribute a file to L receivers. Let the uplink (downlink) bandwidth of receiver i be u_i (d_i, respectively), for $i = 1, \cdots, L$. Let the uplink capacity of the source be u_s. Then, the maximum distribution speed is shown to be

$$\min(u_s, \min_{1 \le i \le L} d_i, \frac{u_s + \sum_{1 \le i \le L} u_i}{L}). \quad (29)$$

In (29), the three terms are the optimal speeds when the bottleneck is at the source upload link, at a download link, or due to the aggregate upload bandwidth, respectively.

By two-phase distribution, we mean each distribution tree has a depth at most 2. The following fact is known to be true.

Fact 3: In a network with only access-speed constraint, the two-phase distribution achieves the (overlay-network) routing capacity [25], which is (29).

In the single source case, there is a maximum flow from the source to each receiver. The minimum of all these maximum flows will be called the *max-flow limit* (MFL). The max-flow limit is a throughput (total distribution rate) upper bound. If all nodes but the source are receivers, the max-flow limit is achievable. This is known as Edmond's Theorem [28] [29].

A. Bottleneck at the Access Links (Profiles 1 to 4)

In profiles 1 to 4, the network bottlenecks are at the access links. Since this is the less interesting case for us, we will only briefly describe the results.

- Our gradient algorithm obtains near the theoretically optimal solution. The gradient algorithm typically uses three kinds of trees, the depth-first search tree (DFS) (a bit surprise), the breadth-first search tree (BFS), and some two-phase trees.
- BitTorrent's performance is not bad. This is well explained in [30].
- AFR's two-phase approach achieves the optimal downloading time when the bottleneck is either at the download link or at the source. But when the bottleneck is due to the aggregate upload bandwidth, AFR fails to achieve the optimum, although we know that some other two-phase solution is optimal. Nevertheless, AFR achieves good performance in this kind of access-limited situation.

B. Bottleneck at the Internal of ISP Backbone (Profile 5)

In this case, we assume all access links have unlimited bandwidth, and congestion happens at the core network. We wish to see how our algorithm performs in infrastructure-mode content distribution. We did experiments with the ISP Sprintlink's backbone obtained from [23]. The network has 315 backbone nodes and 1944 links. It is the largest backbone ISP with the highest node degree among the six commercial backbone networks that the RocketFuel project provides. We attach 100 peers with unlimited access bandwidth randomly to some backbone nodes, with at most one peer per backbone node. One may think a peer is a large content distribution server cluster. Among the 100 peers, we choose 2 sources with the normalized sending rates $r_s = 1.0$ (dimensionless value). Since the actual link capacity data is unavailable, we assign the same bandwidth, 1000, to all backbone links. Then, we scale up the bandwidth of all critical links (those links that, when removed, will leave some peers disconnected in the overlay) to be large enough so that they are not the bottleneck.

We did three tests on Profile 5.

- (Test a) Our algorithm is deployed at all links. This is the case of optimally-allocated overlay bandwidth.
- (Test b) Our algorithm is deployed only at the peers. The overlay bandwidth between each pair of peers is fixed by the max-min allocation. The 100 peers form a single overlay network.
- (Test c) Our algorithm is deployed only at the peers. In order to compare with AFR, we partition the backbone into two overlay networks, one for each source. Note that the max-flow limit is achievable in this case.

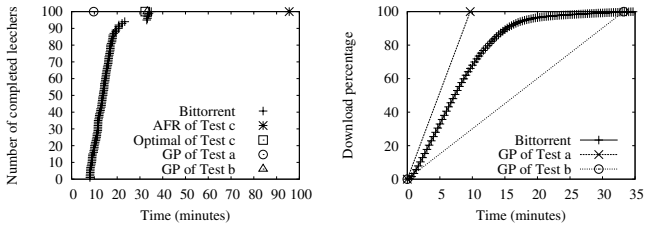


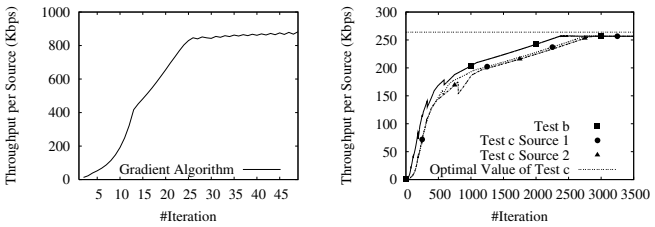
Fig. 3. Profile 5. (a) Number of receivers that have completed download over time; (b) download percentage over time.

1) Comparison with BitTorrent: Fig. 3 shows that, in Test a, the downloading time in the gradient algorithm is only 30% of that in BitTorrent. BitTorrent is unable to give good performance when the core network is congested. But in Test b, after the overlay bandwidth is fixed, the optimal downloading time is much higher than that of Test a, and almost equal to BitTorrent's time. In the figure, the download percentage refers to the total amount of data downloaded at each time instance normalized against the total data downloaded at the end of the distribution. Since the lines have different slopes, we can extrapolate the lines and expect the gradient algorithm to do much better if the file size becomes larger.

2) Comparison with AFR: Fig. 3 also shows that, in Test c, the gradient algorithm approaches the max-flow limit while AFR achieves something far from the optimum. When the congestion happens at the core network, the two-phase trees alone fail to give a good solution.

3) Convergence Speed: Fig. 4 shows the convergence of the algorithm in Test a, b and c. It seems that the algorithm that optimally allocates the overlay bandwidth converges much faster than the algorithm with fixed overlay bandwidth. This has to do with the fact that, in the final solution, Test a has 92 active trees, while Test b has totally 4746 active trees. It is

possible Test b has another optimal or nearly optimal solution that has much fewer trees. Finding solutions with fewer trees should improve convergence speed, and is an important future direction to pursue.



(a) (b)

Fig. 4. Profile 5. Convergence of throughput. Two sources with $r_s = 1.0$ (a) Test a; (b) Test b and c.

C. Bottleneck at the Cross-ISP Links (Profile 6-7)

In reality, the cross-ISP links are often the bandwidth bottleneck. The goal here is to evaluate the effectiveness of our algorithm in handling the bottleneck at cross-ISP links when it is deployed at ISP gateways. We did experiments both on an artificial small cross-ISP network and the cross-ISP network obtained from the RocketFuel project. We created scenarios where congestion happens at the cross-ISP links. Our algorithm is deployed at all access links and cross-ISP links. Hence, we're able to run the algorithm to optimally allocate the overlay bandwidth.

- Profile 6 (P6): 6 completely connected ISPs with 30 cross-ISP links. Each cross-ISP link has a capacity of 1000. 300 peers are attached to the ISPs, 50 per ISP, with sufficient bandwidth. A single source is attached to one ISP.
- Profile 7 (P7): (RocketFuel topology): 69 ISPs connected with 1336 links. The cross-ISP link capacities are uniformly distributed on $(100, 1000)$. 500 peers are randomly attached to the ISPs with sufficient bandwidth. A single source is attached to one ISP.

Note that, in the case of a single source, congestion at the cross-ISP links and each ISP containing some peers, the max-flow limit is achievable.

TABLE I
DOWNLOADING TIME (MINUTES) COMPARISON OF PROFILE 6 AND 7

	BT(50%)	BT(95%)	BT(100%)	GP	AFR	MFL
P6	16	19.5	19.6	3.8	142.7	3.4
P7	5.83	26.5	90	8.74	131.2	8.72

In both profiles 6 and 7, the gradient algorithm approaches the max-flow limit (MFL) and beats BitTorrent and AFR by a large amount, up to a factor of 10 (Tabel I; 50%, 95% and 100% are the percentage of the peers that have finished downloading.). We found, with more peers per ISP, BitTorrent's performance deteriorates. FastReplica's performance is far worse than both the gradient algorithm and BitTorrent.

Usually, cross-ISP traffic is more expensive. We investigated the traffic redundancy over the cross-ISP links. With neither inside ISP congestion nor access speed constraint, ideally, each destination ISP should receive only one copy of each chunk from other ISPs and the source ISP should not receive any copy from other ISPs. In Profile 6, we inspected the active (optimal) trees the algorithm constructed and found that each destination ISP indeed only received one copy from other ISPs, but the source ISP might receive some copies from other ISPs. Suppose the normalized cross-ISP traffic under the ideal distribution is 1.0. We found the cross-ISP traffic was 1.103 in the gradient algorithm and 5.982 in BitTorrent. But in Profile 7, we found the destination ISPs received multiple copies from other ISPs in the gradient algorithm. Again, if the normalized cross-ISP traffic under the ideal distribution is 1.0, then the traffic is 2.22 in the gradient algorithm and 9.72 in BitTorrent.

D. Arrival and Departure Dynamics (Profile 8)

Here, we wish to examine how well the distributed gradient algorithm copes with the peer arrival and departure dynamics. We applied the algorithm with optimally allocated overlay bandwidth on a small star network with receivers arrive and depart randomly. All peers have sufficient download capacities; the receivers each have upload bandwidth 200 Kbps; and the source has upload bandwidth 640 Kbps. In Phase-I, we have one source and 10 receivers; in Phase-II, 2 receivers leave; in Phase-III, 5 new receivers arrive. Assume at the beginning of Phase-III the source has distributed half of the chunks to the 8 existing receivers. We assume, at the end of Phase-III, all 13 receivers finish at the same time (though it's unfair). Thus, we set up two more sessions at Phase-III: one consists of the 5 new receivers and the original source for downloading the second half of the chunks; and the other consists of the 5 new receivers, the source and the 8 existing receivers (now also serving as sources) for downloading the first half of the chunks. Thus, in Phase-III, we have 3 sessions, and in the 3rd session, we have multiple sources with overlapping chunks. Fig. 5 shows the algorithm adapts to the dynamics quickly.

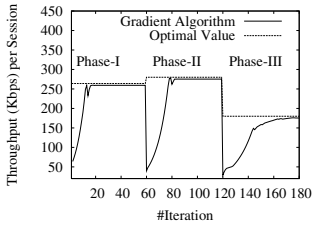


Fig. 5. Dynamic departure and arrival of receivers

V. ADDITIONAL RELATED WORKS

In our optimization-based approach, we follow the tradition of Kelly et. al. [31] and Low et. al. [32] on optimal flow control/bandwidth allocation. Many recent papers extended this approach and solved networking problems by collective actions taken across networking layers, especially in wireless

networks e.g., [33], [34], [35], [36]. Several other related studies, either in topics or methods, are [37], [38], [39], [40].

VI. Conclusion

This paper represents a systematic study on how best to conduct content distribution using advanced P2P techniques. In response to growing massive content that threatens to congest the core network, our objective is to manage the network congestion not only at the access links but throughout the network, especially at cross-ISP links. We showed that this objective is "equivalent" to speeding up content distribution. The main contribution of this paper is that we envision optimal content distribution as a multicast tree packing problem, and we derive a distributed algorithm for solving the problem. The tree-packing framework is also useful for contemplating existing P2P swarming/collaborative downloading techniques, by asking the questions: What kind of trees do existing algorithms use? How are the trees selected? And how is bandwidth assigned to the trees? Hence, our framework has the potential to provide a unified understanding of P2P distribution techniques. Finally, our distributed algorithm is based on a specialized gradient projection algorithm for optimization under simplex constraints and we develop a scaled version of it. Our computation experiences show that it has much faster convergence than the more frequently used subgradient algorithm.

References

[1] K. Cho, K. Fukuda, H. Esaki, and A. Kato, "The impact and implications of the growth in residential user-to-user traffic," in *Proceedings of ACM SIGCOMM*, Pisa, Italy, September 2006.

[2] J. Cannons, R. Dougherty, C. Freiling, and K. Zeger, "Network routing capacity," *IEEE Transactions on Information Theory*, vol. 52, no. 3, pp. 777–788, March 2006.

[3] BitTorrent Web Site, http://www.bittorrent.com/.

[4] J. Lee and G. de Veciana, "On application-level load balancing in FastReplica," *Computer Communications*, to appear, http://users.ece.utexas.edu/~gustavo/papers/LeD07.pdf.

[5] D. Kostić, R. Braud, C. Killian, E. Vandekieft, J. W. Anderson, A. C. Snoeren, and A. Vahdat, "Maintaining high bandwidth under dynamic network conditions," in *Proceedings of USENIX Annual Technical Conference*, 2005.

[6] B. Chun, P. Wu, H. Weatherspoon, and J. Kubiatowicz, "Chunkcast: An anycast service for large content distribution," in *Proceedings of the Internaltional Workshop on Peer-to-Peer Systems (IPTPS)*, February 2006.

[7] D. Bickson, D. Malkhi, and D. Rabinowitz, "Efficient large scale content distribution," in *Proceedings of the 6th Workshop on Distributed Data and Structures (WDAS'2004)*, Lausanne, Switzerland, July 2004.

[8] S. Q. Zhuang, B. Y. Zhao, A. D. Joseph, R. H. Katz, and J. Kubiatowicz, "Bayeux: An architecture for scalable and fault-tolerant wide-area data dissemination," in *Proceedings of the 11th International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV)* , June 2001.

[9] S. Banerjee, B. Bhattacharjee, and C. Kommareddy, "Scalable application layer multicast," in *Proceedings of ACM SIGCOMM 2002*, August 2002.

[10] Akamai Website, http://www.akamai.com.

[11] K. Park and V. S. Pai, "Scale and performance in the coblitz large-file distribution service," in *Proceedings of the 3rd USENIX/ACM Symposium on Networked Systems Design and Implementation (NSDI 06)*, San Jose, CA, May 2006.

[12] M. Grotchel, A. Martin, and R. Weismantel, "Packing Steiner trees: a cutting plane algorithm and computation," *Mathematical Programming*, vol. 72, pp. 125–145, 1996.

[13] K. Jain, M. Mahdian, and M. R. Salavatipour, "Packing Steiner trees," in *Proceedings of the fourteenth annual ACM-SIAM symposium on Discrete algorithms (SODA '03)*, 2003.

[14] D. Bertsekas and R. Gallager, *Data Networks*, 2nd ed. Prentice Hall, 1991.

[15] D. Bertsekas, *Nonlinear Programming*, 2nd ed. Athena Scientific, 1999.

[16] R. Madan, Z. Luo, and S. Lall, "A distributed algorithm with linear convergence for maximum lifetime routing in wireless networks," in *Proceedings of the Allerton Conference on Communication, Control, and Computing*, September 2005.

[17] D. Bertsekas and J. N. Tsitsiklis, *Parallel and Distributed Computation: Numerical Method*. Athena Scientific, 1997.

[18] R. E. Tarjan, "Finding optimum branchings," *Networks, Vol. 7*, pp. 25–35, 1977.

[19] P. M. Camerini, L. Fratta, and F. Maffioli, "A note on finding optimum branchings," *Networks, Vol. 9*, pp. 309–312, 1979.

[20] P. A. Humblet, "A distributed algorithm for minimum weight directed spanning trees," *IEEE Transactions on Communications*, pp. 756–762, June 1983.

[21] Z.-Q. Luo and P. Tseng, "On the rate of convergence of a distributed asynchronous routing algorithm," *IEEE Transcations on Automatic Control*, vol. 39, pp. 1123–1129, May 1994.

[22] R. G. Gallager, "A minimum delay routing algorithm using distributed computation," *IEEE Transactions on Communications*, pp. 73–85, January 1977.

[23] University of Washington, http://www.cs.washington.edu/research/networking/rocketfuel/.

[24] A. R. Bharambe and C. Herley, "Analyzing and improving BitTorrent performance," Microsoft Research, Tech. Rep. MSR-TR-2005-03, 2005.

[25] R. Kumar and K. Ross, "Peer-assisted file distribution: The minimum distribution time," in *IEEE Workshop on Hot Topics in Web Systems and Technologies (HOTWEB)*, 2006.

[26] D. M. Chiu, R. W. Yeung, J. Huang, and B. Fan, "Can network coding help in P2P networks?" in *The fourth International Symposium on Modeling and Optimization in Mobile, Ad Hoc and Wireless Networks*, 2006.

[27] J. Mundinger, R. R. Weber, and G. Weiss, "Optimal scheduling of peer-to-peer file dissemination," *arXiv e-print service*, June 2006, http://arxiv.org/abs/cs.NI/0606110.

[28] Y. Wu, P. A. Chou, and K. Jain, "A comparison of network coding and tree packing," in *The Proceedings of IEEE International Symposium on Information Theory (ISIT)*, June 2004.

[29] J. Edmonds, "Edge-disjoint branchings," in *Combinatorical Algorithms*, R. Rustin, Ed. Academic Press, 1973, pp. 91–96.

[30] D. Qiu and R. Srikant, "Modeling and performance analysis of BitTorrent-like peer-to-peer networks," in *SIGCOMM'04*, 2004.

[31] F. Kelly, A. Maulloo, and D. Tan, "Rate control for communication networks: shadow price, proportional fairness and stability," *Journal of the Operational Research Society*, vol. 49, pp. 237–252, 1998.

[32] S. H. Low and D. E. Lapsley, "Optimization flow control - I: Basic algorithm and convergence," *IEEE/ACM Transactions on Networking*, vol. 7, no. 6, pp. 861–874, 1999.

[33] X. Lin and N. B. Shroff, "Joint rate control and scheduling in multihop wireless networks," in *Proceedings of the 43rd IEEE CDC*, 2004.

[34] A. Eryilmaz and R. Srikant, "Fair resource allocation in wireless networks using queue-length based scheduling and congestion control," in *Proceedings of the IEEE Infocom 2005*, Miami, USA, March 2005.

[35] J. Wang, L. Li, S. H. Low, and J. C. Doyle, "Cross-layer optimization in TCP/IP networks," *IEEE/ACM Transactions on Networking*, vol. 13, no. 3, pp. 582 – 595, June 2005.

[36] L. Chen, S. H. Low, M. Chiang, and J. C. Doyle, "Cross-layer congestion control, routing and scheduling design in ad hoc wireless networks," in *Proceedings of the IEEE Infocom 2006*, Barcelona, Spain, April 2006.

[37] Y. Wu and S.-Y. Kung, "Distributed utility maximization for network coding based multicasting: a shortest path approach," *IEEE Journal on Selected Areas in Communications*, vol. 24, no. 8, pp. 1475– 1488, Aug. 2006.

[38] P. Key, L. Massouliè, and D. Towsley, "Path selection and multipath congestion control," in *Proceedings of INFOCOM 2007*, May 2007.

[39] R. B. et. al., "Improving traffic locality in BitTorrent via biased neighbor selection," in *Proceedings of the International Conference on Distributed Computing Systems (ICDCS'06)*, 2006.

[40] H. Zhang, G. Neglia, D. Towsley, and G. L. Presti, "On unstructured file sharing networks," in *Proceedings of INFOCOM 2007*, May 2007.