Optimal Swarming 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 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*—Content Distribution, Peer-to-Peer Networks, Multicast, Optimization, Bandwidth Allocation**

I. Introduction

One of the distinct trends is that the Internet is being used to transfer content on a more and more massive scale. This has made capacity shortage in the backbone networks a genuine possibility, which will become more serious with fiber-based access[1]. As an example, 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 [3].

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. The proposed solution is a class of improved swarming techniques, known as *optimal swarming*. Swarming was originally invented by the user and research community as a technique for large-scale peer-to-peer file sharing, e.g., [4]–[10]. 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 in a fashion that the sets of chunks at different peers are substantially different. Subsequently, the peers can exchange the chunks with each other to speed up the distribution process. Swarming enables content providers with poor capacity to reach a large number of audience, allows rapid deployment of a large distribution system with minimum infrastructure support.

In this paper, swarming is viewed not just as a casual technique for end-user file-sharing applications. Instead, we envision it as an advanced technique to be employed by critical content distribution applications or other network services in infrastructure networks, such as content distribution networks or ISP networks. We will see that the benefits of swarming rest upon the fact it is an advanced form of networking mechanism, more advanced and more powerful than all previous distribution schemes, including IP or application-level multicast (e.g. [11]), network cache systems and existing content distribution networks (e.g., Akamai [12]). We will argue that 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.

The earlier description of swarming does not specify the precise manner at which the chunks are initially spread out or the manner at which the peers exchange them later. In fact, many different ways of swarming have been proposed, such as BitTorrent [4], FastReplica [5], [6], Bullet [7], [8], Chunkcast [9], CoBlitz [10], and Julia [13]. The most popular one among them is the BitTorrent protocol. The optimal swarming technique proposed in the paper can be contrasted with the existing swarming techniques. First, the optimal swarming is specifically designed for infrastructure networks and its performance objective is network-centric, i.e., how to minimize the worst-case network congestion. [2] In contrast, being originally designed for end-system file-sharing applications, most existing swarming systems have user-centric performance objectives, such as how to complete individual download fast. Second, in existing systems, the bandwidth bottleneck is often assumed to be at the access links, rather than throughout the network. The optimal swarming will be able to automatically adapt to capacity constraint anywhere in the network. Third, most existing systems use heuristic techniques that are not understood well enough regarding how well they work or how much improvement remains possible. On the other hand, the optimal swarming is designed systematically based on the optimization theory and algorithms. As a result, it achieves the best performance with respect to the our chosen objective. The optimization framework not only supplies sophisticated algorithms, which are very difficult to re-invent, it also gives performance guarantees about the algorithms, for instance, in terms of algorithm convergence speed.

For illustration of the main ideas in this paper, consider the toy example in Fig. 1. The numbers associated with the links

[1] At the present, telecom companies are aggressively rolling out fiber-to-the home (FTTH) or its variants. Verizon is building a nationwide FTTH network in the US, to be completed by 2010. Japan had 5.6 million FTTH subscribers by June, 2006, is on an exponential upward ramp to have 30 million by 2010 [1]. The speed of the access fiber is currently at 100 Mbps or lower, heading to 1 Gbps by 2020 and is likely to reach 10 Gbps thereafter [2].

[2] It will be shown later that minimizing the worse-case link congestion is equivalent to minimizing the distribution time of a file, or maximizing the distribution throughput. The equivalence is in the sense that one can recover the solution with respect to one objective from the solution under another objective. In the subsequent discussion, we will use these three objectives interchangeably.

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. 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.

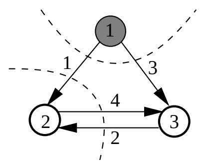


Fig. 1. Node 1 sends the file to node 2 and 3.

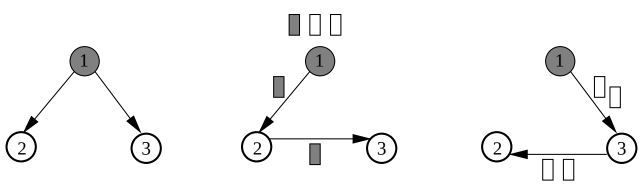


Fig. 2. All possible distribution trees for the example in Fig. 1.

The ideas contained in the toy example are also given in [14]. 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.

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 IV, we discuss practical issues in applying our algorithm to realistic settings, such as scalability and coping with network dynamics and churn. In Section V, we evaluate the performance of our algorithm, including a comparison with BitTorrent and FastReplica. In Section VI, we discuss additional related work. The conclusion is drawn in Section VII.

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 IV-A.) 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,

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

The optimization problem is as follows.

$$\min \quad t \tag{1}$$

$$\text{s.t.} \quad t \sum_{i=1}^{|T_s^{(m)}|} z_{s,i}^{(m)} = L_s^{(m)}, \quad \forall s \in S^{(m)},\ \forall m \in M \tag{2}$$

$$x_e \leq c_e, \quad \forall e \in E \tag{3}$$

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

Condition (2) 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 travel along exactly one tree (and hence, be received by each receiver exactly once). A moment of thinking reveals that nothing is gained by sending the bit on more than one tree. Condition (3) 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 (2), $z_s^{(m)}$ 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 $t = \mu L_s^{(m)}/r_s^{(m)}$, and redefine $z_{s,i}^{(m)}$ to be $\mu z_{s,i}^{(m)}$. Now, x_e is the aggregate link flow rate of link e under the redefined tree rates. Then, we get the following minimizing-congestion formulation.

$$\begin{aligned} \min \quad & \mu \qquad (4)\\ \text{s.t.} \quad & \sum_{i=1}^{|T_s^{(m)}|} z_{s,i}^{(m)} = r_s^{(m)}, \quad \forall s \in S^{(m)},\ \forall m \in M \\ & x_e \le \mu c_e, \quad \forall e \in E \\ & z_{s,i}^{(m)} \ge 0, \quad \forall i = 1, \cdots, |T_s^{(m)}|, \forall s \in S^{(m)}, \forall m \in M. \end{aligned}$$

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. Let $(\vec{z}^*, \vec{x}^*, \vec{\mu}^*)$ be an optimal solution of the congestion minimization problem (4). Then, $(\vec{z}^*/||\vec{\mu}^*||_\infty,\ \vec{x}^*/||\vec{\mu}^*||_\infty)$ is optimal to the original problem (1). In the minimization problem (4), $\vec{z}$ and $\vec{x}$ are redefined as the vector of tree rates and the vector of aggregate link rates under the fixed demand rate vector $\vec{r}$, respectively.

Thus, we have two equivalent views of optimal multicast tree packing. In the first, the objective is to minimize the overall distribution time (or maximize the distribution throughput) while satisfying the link capacity constraint. In the second, 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.

$$\min \quad ||\vec{\mu}||_\infty \qquad (5)$$

$$\text{s.t.} \quad \sum_{i=1}^{|T_s^{(m)}|} z_{s,i}^{(m)} = r_s^{(m)}, \quad \forall s \in S^{(m)},\ \forall m \in M \qquad (6)$$

$$x_e = \mu_e c_e, \quad \forall e \in E \qquad (7)$$

$$z_{s,i}^{(m)} \ge 0, \quad \forall i = 1, \cdots, |T_s^{(m)}|, \forall s \in S^{(m)}, \forall m \in M. \qquad (8)$$

The optimization problem proposed so far is equivalent to the problem of packing Steiner trees [15], [16], which is computationally intractable. Fortunately, for content distribution on overlay networks, the problem becomes simpler. For each session, an overlay network is constructed over exactly those nodes (members) in the session. For any node i and node j in the overlay network, there is a directed overlay link from node i to node j. This overlay link is in fact the path in the underlay network from node i to node j, which contains a known set of physical links. Hence, the overlay network is directed and fully connected. Given an underlay network and $|M|$ distribution sessions, $|M|$ such overlay networks are constructed. However, at this point, the bandwidth of each overlay link has not been determined. We will consider two possibilities about how to determine the overlay bandwidth in Section II-B and II-C.

Since the overlay network of each session consists of exactly those nodes in the session, any Steiner tree that covers all nodes of the session is in fact a spanning tree. We will show later that our optimization algorithm involves a minimum-cost spanning tree subproblem 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.

B. Fixed Overlay Link Bandwidth

In this case, the bandwidth of each overlay link is a fixed constant[3] 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 [17]. 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

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

The optimization problem is now

$$\begin{aligned} & \min ||\vec{\mu}||_\infty \qquad (9)\\ & \text{s.t.} \sum_{i=1}^{|T_s|} z_{s,i} = r_s, \quad \forall s \in S \\ & x_e = \mu_e c_e, \quad \forall e \in \hat{E} \\ & z_{s,i} \ge 0, \quad \forall i = 1, \cdots, |T_s|, \quad \forall s \in S. \end{aligned}$$

[3]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.

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 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

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

The optimization problem is exactly written as in (5)-(8).

III. Distributed Algorithm: Diagonally Scaled Gradient Projection

Note that $\min ||\vec{\mu}||_\infty$ has the same solution as $\min ||\vec{\mu} + \vec{\kappa}||_\infty$, where $\vec{\kappa} = (\kappa, \ldots, \kappa)$ for some small constant $\kappa \geq 0$. We replace the objective function $||\vec{\mu}||_\infty$ by $||\vec{\mu} + \vec{\kappa}||_\infty$ in (5) and keep the same constraints. $\vec{\kappa}$ serves as a regularization term. The strictly positive vector $\vec{\kappa}$ (i.e., $\kappa > 0$) guarantees that our gradient projection algorithm has a global geometric convergence rate; if $\vec{\kappa} = \vec{0}$, we can only claim that our algorithm converges to one optimal solution globally.

A. Fixed Overlay Link Bandwidth

The goal of minimizing $||\vec{\mu}+\vec{\kappa}||_\infty$ 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 $||\vec{\mu} + \vec{\kappa}||_q = (\sum_{e\in\hat{E}}(\mu_e+\kappa)^q)^{1/q}$. In fact, $||\vec{\mu}+\vec{\kappa}||_\infty$ can be approximated by $||\vec{\mu} + \vec{\kappa}||_q$: as $q \to \infty$, $||\vec{\mu} + \vec{\kappa}||_q \to ||\vec{\mu} + \vec{\kappa}||_\infty$. We will assume $q \geq 2$ throughout. Since $\min ||\vec{\mu} + \vec{\kappa}||_q$ is equivalent to $\min ||\vec{\mu}+\vec{\kappa}||_q^q$, after a substitution of μ_e with x_e/c_e, we get an approximation of problem (9) as

$$\min \sum_{e\in\hat{E}} (\frac{x_e}{c_e} + \kappa)^q \tag{10}$$

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

$$z_{s,i} \geq 0, \quad \forall i = 1, \cdots, |T_s|, \quad \forall s \in S. \tag{12}$$

For optimization problems with the simplex constraint (11), the optimality condition is especially simple [18]. It has been shown in [19] and [20] 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 one 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 (10), let the feasible set defined by (11) and (12) 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 $\vec{x}$ 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 $\hat{f}_e(x_e) = (x_e/c_e + \kappa)^q$ and $\hat{f}(x) = \sum_{e\in\hat{E}} \hat{f}_e(x_e)$. The objective function, denoted by $f(z)$, is given by

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

and (10) can be written as

$$\begin{aligned} \min \quad & f(z) = \hat{f}(Hz) \\ \text{s.t.} \quad & z \in \mathcal{Z}. \end{aligned} \tag{13}$$

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

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

Note that $\frac{\partial \hat{f}(x_e)}{\partial x_e}$ is the so-called first-derivative link cost of link e [18], [20]. 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. The second derivative of $f(z)$ will be used in the scaling of the algorithm. With respect to $z_{s,i}$ and $z_{s,j}$, it is given by

$$\frac{\partial^2 f(z)}{\partial z_{s,i} \partial z_{s,j}} = \sum_{e\in t_{s,i}\cap t_{s,j}} \frac{q(q-1)}{c_e^2} (\frac{x_e}{c_e} + \kappa)^{q-2}. \tag{14}$$

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,

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

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 (10) by the following optimality condition.

$$\sum_{s \in S} \sum_{i: t_{s,i} \in T_s} \frac{\partial f(z^*)}{\partial z_{s,i}} (z_{s,i} - z^*_{s,i}) \geq 0, \ \forall z \in \mathcal{Z}. \tag{15}$$

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

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

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 will develop the gradient projection algorithm following the proposal in [20] to solve the problem (10). But we will add diagonal scaling to speed up the algorithm's convergence time.

The equality constraint in (11) implies that, for each source, one of the variables depends completely on the rest of the variables. We can eliminate this variable and have a problem with fewer variables. To be concrete, at a feasible vector z, let's eliminate the variable z_{s,i_s} for each $s \in S$. Define a new objective function $g(\hat{z})$ on $\mathbb{R}^{|T|-|S|}$, where $\hat{z}$ consists of the remaining $z_{s,i}$'s after z_{s,i_s} is eliminated for each s. Without loss of generality, suppose, for each source s, i_s corresponds to the tree with the largest index, i.e., $i_s = |T_s|$. Also suppose the sources are index from 1 to $|S|$. Then,

$$\begin{aligned}\hat{z} =&(z_{1,1}, \ldots, z_{1,|T_1|-1}; z_{2,1}, \ldots, z_{2,|T_2|-1}; \\ &\ldots; z_{|S|,1}, \ldots, z_{|S|,|T_{|S|}}).\end{aligned}$$

We will call the domain of g where $\hat{z}$ lies the reduced domain. We let

$$\begin{aligned}g(\hat{z}) =& f(z_{1,1}, \ldots, z_{1,|T_1|-1}, r_1 - \sum_{j=1}^{|T_1|-1} z_{1,j}; \\ & z_{2,1}, \ldots, z_{2,|T_2|-1}, r_2 - \sum_{j=1}^{|T_2|-1} z_{2,j}; \quad \ldots; \\ & z_{|S|,1}, \ldots, z_{|S|,|T_{|S|}-1|}, r_{|S|} - \sum_{j=1}^{|T_{|S|}|-1} z_{|S|,j}).\end{aligned}$$

The optimization problem in (10)- (12) is equivalent to

$$\min_{\hat{z} \geq 0} g(\hat{z}).$$

This problem can be solved by the gradient projection algorithm.

$$\hat{z}(k+1) = [\hat{z}(k) - \delta(k) \nabla g(\hat{z}(k))]_+, \tag{16}$$

where $\delta(k)$ is a positive step size and $[\ ]_+$ is the projection operator on $\hat{z} \geq 0$. In this case, $[y]_+$ just means that, if y_i is a component of y, we take $\max(y_i, 0)$ as the corresponding component of the vector $[y]_+$. The key is to compute $\nabla g(\hat{z}(k))$. Let $i_s(k)$ be a short hand for $i_s(z(k))$. It is easy to show, for $s \in S$ and $i \neq i_s(k)$,

$$\frac{\partial g(\hat{z}(k))}{\partial z_{s,i}} = \frac{\partial f(z(k))}{\partial z_{s,i}} - \frac{\partial f(z(k))}{\partial z_{s,i_s(k)}}.$$

The first derivatives are given in (14).

The algorithm in (16) is actually the constrained steepest-descent algorithm. It is well-known that the steepest-descent algorithm can be slow if the optimization problem is ill-conditioned. 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. An ultimate solution to an ill-conditioned problem is Newton's algorithm. However, Newton's algorithm is generally very complex since it requires the inverse of the Hessian matrix of the objective function. For large problems, this computation is generally impractical. We will next develop the diagonally scaled gradient algorithm, which is a much simpler alternative and a good approximation of Newton's algorithm. The scaled gradient projection algorithm can be written as

$$\hat{z}(k+1) = [\hat{z}(k) - \delta(k) D(k) \nabla g(\hat{z}(k))]_+,$$

where $D(k)$ is a positive definite matrix. For diagonal scaling, $D(k)$ is chosen to be a diagonal matrix.

$$D(k) = \text{diag}[(d_{s,i}(k))^{-1}]_{s \in S, i \neq i_s(k)}.$$

That is, the diagonal entry corresponding to $z_{s,i}(k)$ is chosen to be $(d_{s,i}(k))^{-1}$. For each $s \in S$ and $i \neq i_s(k)$, the value of $d_{s,i}(k)$ is chosen to be

$$d_{s,i}(k) = \frac{\partial^2 g(\hat{z}(k))}{\partial z^2_{s,i}}.$$

This way, the matrix $D(k)$ approximates the inverse of the Hessian of g at $\hat{z}(k)$. For each $s \in S$ and $i \neq i_s(k)$, the second derivative of g is further given by

$$\frac{\partial^2 g(\hat{z}(k))}{\partial z^2_{s,i}} = \frac{\partial^2 f(z(k))}{\partial z^2_{s,i}} + \frac{\partial^2 f(z(k))}{\partial z^2_{s,i_s(k)}} - 2 \frac{\partial^2 f(z(k))}{\partial z_{s,i} \partial z_{s,i_s(k)}}.$$

The second derivatives are given in (14).

We can now collect different pieces of the development above and formally give the diagonally scaled gradient projection algorithm in the original domain where z lies. A slight generalization is present in (17).

Diagonally Scaled Gradient Projection Algorithm

$$z(k+1) = \alpha(k) \bar{z}(k) + (1 - \alpha(k)) z(k) \tag{17}$$

$$\bar{z}_{s,i}(k) = \tag{18}$$

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

with

$$d_{s,i}(k) = \sum_{e \in t_{s,i} \cup t_{s,i_s(k)} \setminus t_{s,i} \cap t_{s,i_s(k)}} \frac{q(q-1)}{c_e^2} (\frac{x_e(k)}{c_e} + \kappa)^{q-2}. \tag{19}$$

In (17), $\alpha(k)$ is a scalar on $[\underline{a}, 1]$, for some $\underline{a}$, $0 < \underline{a} \leq 1$. (17) 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 (18), 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 (more precisely, if $t_{s,i}$ is a minimum-cost tree with positive flow rate but not the chosen minimum-cost tree, its rate will keep the same).

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 (17), 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. [4]

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 (18), the rate reduction is proportional to this difference with a proportional constant (step size) $\delta_s(k) > 0$.

The factor $(d_{s,i}(k))^{-1}$ does diagonal scaling, which can effectively deal with our ill-conditioned problem. Scaling by $(d_{s,i}(k))^{-1}$ can be understood as allowing different components of the vector z to use different step sizes. Note that the expression for $d_{s,i}(k)$ in (19) corresponds to the i^{th} tree; the sum is over the non-overlapping links between the i^{th} tree and the $i_s(k)^{th}$ tree, the latter being the minimum-cost tree.

The algorithm in (17)-(19) is a distributed one. In order to compute the tree cost, $\frac{\partial f(z)}{\partial z_{s,i}}$ in (14), and the scaling factor, $(d_{s,i}(k))^{-1}$ in (19), 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 [21] [22] [23]. 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 among the set of 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 (9), 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.

We stress that the reason to apply the scaling factor is to counter the ill-conditioned problem when q is large. Though the steepest-descent gradient projection algorithm has a linear (i.e., geometric) converge rate asymptotically, in practice, it is often slow if the optimization problem is ill-conditioned. Diagonal scaling often significantly outperforms the plain steepest-descent algorithm. In an ill-conditioned problem, single-unit changes of different variables have disproportionate effects on the cost (objective value) change [18]. For convergence, the step size in the iterative algorithm must be tuned according to the variables that cause large cost changes. However, such a step size can be too small for other variables, and as a result, their values hardly change from iteration to iteration. The diagonally scaled algorithm essentially re-scales the variables so that single-unit changes in the scaled variables have similar effect on the cost objective. For our problem, the scaling has the simple interpretation that different trees use different step sizes, each roughly being proportional to a power of the worst link utilization on the tree. Note that by (19), the scaling factor $(d_{s,i}(k))^{-1}$ is roughly inversely related to the cost difference $\frac{\partial f(z(k))}{\partial z_{s,i}} - \frac{\partial f(z(k))}{\partial z_{s,i_s(k)}}$. Thus when the step size is too small for a tree (i.e., the cost difference between the tree and the minimum-cost tree is also small), the scaling factor $(d_{s,i}(k))^{-1}$ will be large and can compensate the small step size. The resulting scaled algorithm is far superior to the plain gradient projection algorithm.

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. Substitute μ_e with x_e/c_e, the problem becomes

$$\min \quad \sum_{e \in E} \left(\frac{x_e}{c_e} + \kappa\right)^q \tag{20}$$

$$\text{s.t.} \quad \sum_{i=1}^{|T_s^{(m)}|} z_{s,i}^{(m)} = r_s^{(m)}, \quad \forall s \in S^{(m)}, \ \forall m \in M \tag{21}$$

$$z_{s,i}^{(m)} \geq 0, \quad \forall i = 1, \cdots, |T_s^{(m)}|, \forall s \in S^{(m)}, \forall m \in M. \tag{22}$$

Let $T = \bigcup_{m \in M}^{s \in S^{(m)}} T_s^{(m)}$ be the collection of all multicast trees rooted at any source s for any session m. Let z be the vector $(z_{s,i}^{(m)})$ where $s \in S^{(m)}, m \in M, i = 1, \ldots, |T_s^{(m)}|$, with an arbitrary indexing order for the sources. In problem (20)-(22), let the feasible set defined by (21) and (22) be denoted by $\mathcal{Z}$.

Let $\hat{E} = \bigcup_{m \in M} \hat{E}^{(m)}$ be the collection of all overlay links in all sessions. Let $\hat{H}$ denote the $|\hat{E}| \times |T|$ overlay link-tree incidence matrix associated with the trees in T (i.e. $[\hat{H}]_{\hat{e}t} = 1$ if overlay link $\hat{e}$ lies on tree t; and $[\hat{H}]_{\hat{e}t} = 0$ otherwise). Recall E is the set of underlay links, let H denote the $|E| \times |\hat{E}|$ underlay link-overlay link incidence matrix associated with the overlay links in $\hat{E}$ (i.e. $[H]_{e\hat{e}} = 1$ if underlay link e lies on overlay link (underlay path) $\hat{e}$; and $[H]_{e\hat{e}} = 0$ otherwise).

For each underlay link $e \in E$, recall that x_e is the aggregate flow rate it carries. Let x be the vector $(x_e)_{e \in E}$. It is easy

[4] (17) can be replaced with a more general update $z(k+1) = A(k)\bar{z}(k) + (I - z(k))z(k)$, where $A(k)$ is a $\sum_{s \in S} |T_s| \times \sum_{s \in S} |T_s|$ diagonal matrix with diagonal entries in the interval $[\underline{a}, 1]$, for some $\underline{a}$, $0 < \underline{a} \leq 1$. To ensure feasibility of $z(k+1)$ in $\mathcal{Z}$, it is required that $\sum_{1 \leq i \leq |T_s|} a_{s,i}(k)(\bar{z}_{s,i}(k) - z_{s,i}(k)) = 0$, where $a_{s,i}(k)$ is a corresponding diagonal entry of $A(k)$.

to see $x = H\hat{H}z$. Note that an underlay link e might carry multiple copies of the same file chunk distributed by one tree t. Define $\hat{f}_e(x_e) = (x_e/c_e + \kappa)^q$ and $\hat{f}(x) = \sum_{e \in E} \hat{f}_e(x_e)$. The objective function, denoted by $f(z)$, is given by

$$f(z) = \hat{f}(H\hat{H}z) = \sum_{e \in E} \hat{f}_e(x_e) = \sum_{e \in E} (\frac{x_e}{c_e} + \kappa)^q,$$

and (20) can be written as

$$\begin{array}{ll} \min & f(z) = \hat{f}(H\hat{H}z) \\ \text{s.t.} & z \in \mathcal{Z}. \end{array}$$

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

$$\begin{aligned} \frac{\partial f(z)}{\partial z_{s,i}^{(m)}} &= \sum_{\hat{e} \in t_{s,i}^{(m)}} \sum_{e \in \hat{e}} \frac{\partial \hat{f}(x_e)}{\partial x_e} \\ &= \sum_{\hat{e} \in t_{s,i}^{(m)}} \sum_{e \in \hat{e}} \frac{q}{c_e} (\frac{x_e}{c_e} + \kappa)^{q-1}. \end{aligned}$$

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

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

Let $i_s^{(m)}(k)$ be a short hand of $i_s^{(m)}(z(k))$.

Diagonally Scaled Gradient Projection Algorithm

$$z(k+1) = \alpha(k)\bar{z}(k) + (1 - \alpha(k))z(k) \tag{23}$$

$$\bar{z}_{s,i}^{(m)}(k) = \tag{24}$$

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

with

$$d_{s,i}^{(m)}(k) = \tag{25}$$

$$\sum_{\hat{e} \in t_{s,i}^{(m)} \cup t_{s,i_s^{(m)}(k)}^{(m)} \setminus t_{s,i}^{(m)} \cap t_{s,i_s^{(m)}(k)}^{(m)}} \sum_{e \in \hat{e}} \frac{q(q-1)}{c_e^2} (\frac{x_e(k)}{c_e} + \kappa)^{q-2}.$$

The resulting algorithm is still fully distributed.

C. Convergence Results

We will show the convergence results of the synchronous gradient projection algorithm under constant step size, i.e., $\delta(k) = \delta$ for all k. We will adapt the results from [24] to find an upper bound on the step size δ that guarantees the global convergence of the synchronous gradient projection algorithm to an optimal solution. Furthermore, with the strictly positive regularization vector $\vec{\kappa}$ (i.e., $\kappa > 0$), the convergence speed is linear (i.e., geometric). The same convergence results can be said for the case of optimally allocated bandwidth.

In the optimization problem (13), we assume $q \geq 2$, so that $\hat{f}_e(x_e)$ is continuous on the interval $[0, \infty)$, tends to ∞ as x_e approaches ∞, and its derivative and second derivative are continuous and positive on $(0, \infty)$. Assuming the links are indexed from 1 to $|\hat{E}|$, the Hessian $\nabla^2 \hat{f} = \text{diag}[\frac{\partial^2 \hat{f}}{\partial x_1^2}, \cdots, \frac{\partial^2 \hat{f}}{\partial x_{|\hat{E}|}^2}]$ is an $|\hat{E}| \times |\hat{E}|$ diagonal matrix with nonnegative diagonal entries. Furthermore, if $\kappa > 0$, the diagonal entries of $\nabla^2 \hat{f}$ are positive and bounded below by $\min_{e \in \hat{E}} \{\frac{q(q-1)}{c_e^2} \kappa^{q-2}\}$.

We assume there is at least one feasible solution, i.e., $z(0) \in \mathcal{Z}$ satisfying $Hz(0) \in \prod_{e \in \hat{E}} [0, \infty)$, and define a compact set $\mathcal{Z}_0 = \{z \in \mathcal{Z} | f(z) \leq f(z(0))\}$. Since this set is compact, f must attain a minimum on this set. Hence, there is a $z^* \in \mathcal{Z}_0$ satisfying $f(z^*) = f^*$, where $f^* = \min_{z \in \mathcal{Z}_0} f(z)$. We call any such z^* an optimal solution. Denote by $\mathcal{Z}^*$ the set of optimal solutions (there may be more than one optimal solutions since, although $\hat{f}$ is strictly convex, f is not), i.e.,

$$\mathcal{Z}^* = \{z \in \mathcal{Z}_0 | f(z) = \min_{z \in \mathcal{Z}_0} f(z)\}.$$

For simplicity, we assume the scaling factor $d_{s,i}(k) = 1.0$. The convergence results still hold for other scaling factors as long as $\{d_{s,i}(k)\}$ are bounded between two fixed positive scalars [24] [18]. Note that $\{d_{s,i}(k)\}$ is bounded below by $\min_{e \in \hat{E}} \{\frac{q(q-1)}{c_e^2} \kappa^{q-2}\}$, and can be bounded above by a fixed constant if the demand rate vector r is carefully chosen so that a feasible solution [5] is achieved after a few steps.

Let $\delta_1 = \underline{a}/(L \max_{s \in S} |T_s|)$, where $L > 0$ is an upper bound of the norm of $\nabla^2 f$ over $\mathcal{Z}_0$.

Lemma 1: For $0 < \delta \leq \delta_1$, we have for all k such that $z(k) \in \mathcal{Z}_0$

$$f(z(k{+}1)) - f(z(k)) \leq -(\frac{\underline{a}}{\delta \max_{s \in S} |T_s|} - \frac{L}{2}) ||z(k) - \bar{z}(k)||^2. \tag{26}$$

The proof of Lemma 1 follows the proof for a similar lemma in [24]. The only change involves substitution of appropriate constants.

Therorem 2: (Globally Convergence) Suppose $\kappa \geq 0$. For any δ, $0 < \delta < \delta_1$, every limit point of $\{z(k)\}$ generated by the synchronous gradient projection algorithm (17)-(19) with $z(0) \in \mathcal{Z}_0$ is optimal.

Proof: With the constant step size $\delta < \frac{\underline{a}}{L \max_{s \in S} |T_s|}$, the right-hand side of the inequality (26) is non-positive. Hence, if $\{z(k)\}$ has a limit point, the left-hand side tends to 0. The algorithm (17)-(19) can be denoted as a function $A(z)$, i.e., $z(k+1) = A(z(k))$. Therefore, $||z(k) - \bar{z}(k)|| \to 0$, which implies that for every limit point $\tilde{z}$ of $\{z(k)\}$ we have $\tilde{z} = A(\tilde{z})$. It is easy to show if $\tilde{z} = A(\tilde{z})$, for any $s \in S$, we have $\tilde{z}_i^s > 0$ only if $\frac{\partial f(\tilde{z})}{\partial \tilde{z}_{s,i}} \leq \frac{\partial f(\tilde{z})}{\partial \tilde{z}_{s,j}}, \forall t_{s,j} \in T_s$, which is exactly the optimality condition in (15). So $\tilde{z}$ is stationary (Proposition 2.3.2 and Example 2.1.2 in [18]). ■

When the regularization vector $\vec{\kappa}$ is strictly positive, the diagonal entries of $\nabla^2 \hat{f}$ are positive and bounded below by $\min_{e \in \hat{E}} \{\frac{q(q-1)}{c_e^2} \kappa^{q-2}\} > 0$. When $q = 2$, the diagonal entries of $\nabla^2 \hat{f}$ are positive and bounded below by $\min_{e \in \hat{E}} \{\frac{q(q-1)}{c_e^2}\} > 0$ for all $\kappa \geq 0$. In these two cases, all conditions for global geometric convergence required by [24] are satisfied. We have a global geometric convergence rate for algorithm (17)-(19).

Therorem 3: (Globally Geometric Convergence Rate)

Suppose $\kappa > 0$. Let δ satisfy $0 < \delta \leq \delta_1$. The sequence $\{z(k)\}$ generated by the synchronous gradient projection algorithm (17)-(19) converges to an element of $\mathcal{Z}^*$ with an initial

[5]A feasible solution means that the aggregate link flow rate is no more than the link capacity for each link.

feasible $z(0)$ and the convergence rate is linear (i.e., geometric) in the sense that for all k,

$$f(z(k+1)) - f^* \leq (1 - D_5\delta)(f(z(k)) - f^*).$$

Furthermore, when $q = 2$, the above conclusion holds for all $\kappa \geq 0$.

The constants and parameters are as follows. $D_5 = \underline{a}/(D_4 + \delta_1)$, $D_4 = ((5L + 1)(D_3)^2 + 1 + 2\delta_1 + 6L(\delta_1)^2/\underline{a}) \max_{s \in S} |T_s|$ and $D_3 = D \max\{1, \delta_1\}$ for some $D > 0$. Moreover, D is bounded above by $D_1(D_1 + (\sqrt{\max_{s \in S} |T_s|} + 1)\hat{L}||H^T||)/\hat{\sigma}$, where $D_1 = \max\{||Q^{-1}|| \mid Q$ an invertible submatrix of $H\}$. $\hat{\sigma} \leq \hat{L}$ are any two positive scalars such that the diagonal entries of $\nabla^2 \hat{f}(Hz)$ lie inside $[\hat{\sigma}, \hat{L}]$ for all $z \in \mathcal{Z}_0$, where $\nabla^2 \hat{f}(Hz)$ is a positive diagonal matrix.

IV. Practical Considerations

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

A. Overlapping Content

We introduce virtual sources to handle the situation where some sources share overlapping (common) chunks. For each group of chunks that exist at multiple sources, we create a virtual source, which will be considered as the source for the overlapping chunks in question. The virtual source has one outgoing virtual link with infinite capacity connecting to each of the above original sources. We then arrive at an expanded network where the sources and virtual sources no longer contain overlapping chunks. If, initially, the way chunks overlap at different original sources is not very complex, the number of resulting virtual sources will be small. Otherwise, we can always neglect some redundant sources to reduce the number of virtual sources. We have conducted a separate study on how to do this systematically.

At each iteration, each virtual source computes the MDSP tree, updates the tree rates and allocates chunks to the active trees. Since the capacity of a virtual link is infinite, its first-derivative link cost must be zero. In the resulting MDSP tree rooted at the virtual source, the virtual source is directly connected to all the original sources that contain the overlapping chunks in question. The virtual source can inform the original sources about the chunk allocation and leave the actual transmission to the original sources. The original sources do not actually receive the chunks from the virtual source, but only the control signals. In actual operation, one of the original physical sources will act as the virtual source and run the algorithm assigned to the virtual source.

B. Mixed Architecture of Fixed and Allocated Bandwidth

In Section II-B and II-C, we see two content distribution scenarios with either fixed or optimally-allocated overlay bandwidth. The latter should achieve better downloading time than the former. However, the latter requires the deployment of our algorithms to all network element, i.e. routers, which is almost certainly impossible. There is an alternative framework in which some routers or devices attached to the routers are deployed with our algorithm, while others not. For instance, our algorithm can be deployed at the cross-ISP links and access links, where the bandwidth is more likely to be small. In this framework, for those physically directly connected router pairs deployed with our algorithms, we model the physical links between them exactly; while for those not directly connected devices, routers and end-systems, we let TCP allocate the end-to-end bandwidth between them and model these end-to-end paths as overlay links. Thus, in our graph of the network, some links are real physical links and others are overlay links with TCP-allocated bandwidth. The algorithm applies as usual.

C. 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 mostly considering the distribution of massive content on managed infrastructure networks, the assumption is reasonable for the most part. However, we do need to deal with low-degree member churn and network dynamics such as link capacity variations and failures.

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 [25], 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 V, we will show a small example of how our algorithm successfully adapts to the departure and arrival of receivers.

D. 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 [23]. But the price to pay is the potentially slower speed due to the coordination overhead of 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. Though currently we don't have a well-defined way to partition the session, we will show one naive approach to partition a large session in Section V.

E. Asynchronous Algorithm

Time synchrony is usually difficult to maintain in a large network. An asynchronous version of the scaled gradient algorithm (17)-(19) (or (23)-(25), respectively) could be developed and the corresponding convergence result could be stated following the approach in [26] [24].

V. Performance Evaluation

In this section, we show performance evaluation results about our optimal swarming algorithm and compare the optimal swarming with other interesting swarming techniques. To fully appreciate these results, it is important to recall that the application setting of our algorithm is infrastructure-based content distribution networks. Unlike the end-system-based P2P file-sharing networks, these infrastructure overlay networks are generally managed, moderate in size (up to tens of thousands of nodes instead of millions), and relatively stable with low node arrival and departure dynamics.

We will compare the performance of our gradient algorithm (GP) with known theoretical bounds, BitTorrent (BT) and Adaptive FastReplica (AFR) [6]. We select BitTorrent because its techniques are interesting and it is very popular in end-system-based distribution. The techniques of BitTorrent can certainly be applied to infrastructure-based distribution and we would like to see how they compare with our algorithm. Since infrastructure-based content distribution is a relatively stable environment, we will do experiments with BitTorrent under low node arrival and departure dynamics. 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.

Although we are more interested in network interior bottlenecks, our algorithm can equally deal with bottlenecks at the access links, at the ISP backbone or at the cross-ISP links. Hence, we will consider all these cases. The commercial ISP backbone and cross-ISP topologies are obtained from the Rocketfuel project [27]. In the terminology of BitTorrent, a seed is a source, and a leecher is a receiver. 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 regulation term κ should be selected small enough so that $\vec{\kappa}$ does not dominate $\vec{\mu}$. Recall that κ is used to guarantee globally geometric convergence. Even if $\kappa = 0$, the proposed algorithm still converges and has a locally geometric convergence rate. In practice, $\kappa = 0$ often works well enough.

A. The Performance Evaluation Metrics

1) BitTorrent Simulation: We use the Bittorrent simulator developed by Bharambe et. al. [28]. 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 [17]. In the BitTorrent simulation, we use the following simulation environment.

- Each peer opens 5 uploading connections and one of them is selected by optimistic unchoking, which means that it connects to a random neighbor.
- The seed uses the smart seed policy, which is introduced in [28]. Seeds are with distinct files.
- All the peers join the network at time 0 and continue to be in the network until all of the leechers complete the download.
- All the other parameters follow the regular BitTorrent environment.

Table I summarizes the simulation environment for our tests.

TABLE I
BitTorrent Simulation Parameters

	#Seeds	File Size per Seed	Neighborhood Size
Profiles 1-3	1	62.765 MB	38 - 80
Profile 4	1	128 MB	18 - 40
Profile 5	2	64 MB	18 - 40
Profile 6	1	128 MB	38 - 80
Profile 7	1	32 MB	38 - 80

2) 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 [6], the theoretical throughput of AFR in its best behavior can be computed as

$$\sum_{i=1,\cdots,m} \min\{c_{n_0 n_i}, \min_{j=1,\cdots,m, j\neq i}\{c_{n_i n_j}\}\}, \tag{27}$$

where n_0 is the source, n_i, $i = 1, \cdots, m$ are the receivers, and $c_{n_i n_j}$ is the end-to-end path (overlay link) capacity between n_i and n_j.

3) Theoretical Bounds: Some theoretical results are used as performance benchmark in our performance comparison. In several studies [29]–[31], researchers have analyzed a model of P2P file sharing among residential users in low access-speed environment. Each participating end-system has an uplink (to the network) and a downlink with limited capacity. The capacity of the network is considered unlimited. 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

$$\min(u_s, \min_{1\leq i\leq L} d_i, \frac{u_s + \sum_{1\leq i\leq L} u_i}{L}). \tag{28}$$

In (28), 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 4: In a network with only access-speed constraint, the two-phase distribution [29] achieves the (overlay-network) routing capacity [29], which is (28).

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, a result known as Edmond's Theorem [32] [33].

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

In this case, we assume the network has infinite capacity but the access links have finite capacities. We also assume that all access links are deployed with our gradient algorithm. In the four test cases (profile 1-4), we have a single source with uploading bandwidth u_s. Let u_i and d_i be leecher (receiver) i's upload and download bandwidth, respectively.

- Profile 1: $u_i = d_i = 360$ Kbps for all 299 receivers, $u_s = 640$ Kbps. The download link is the bottleneck.
- Profile 2: $u_i = d_i = 360$ Kbps for all 299 receivers, $u_s = 280$ Kbps. The source upload link is the bottleneck.
- Profile 3: $d_i = 360$ Kbps, $u_i = 200$ Kbps for all 299 homogeneous receivers, $u_s = 640$ Kbps. The aggregate upload bandwidth is the bottleneck.
- Profile 4: $d_i = 360$ Kbps for all 100 receivers, $u_i = 100$ Kbps for half of receivers, and $u_i = 1$ Kbps for the rest receivers. $u_s = 100$ Kbps. The aggregate upload bandwidth is the bottleneck.

TABLE II
COMPARISON OF DOWNLOADING TIME (MINUTES) AND NUMBER OF ACTIVE TREES.

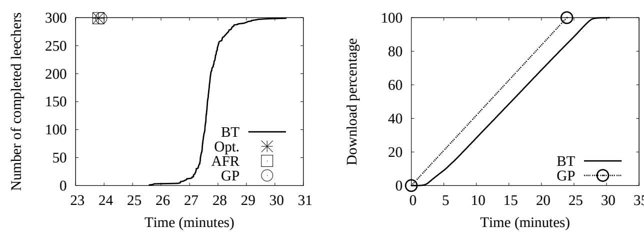
	Profile 1	Profile 2	Profile 3	Profile 4
Optimum	23.8	30.6	42.4	331.4
BT(50%)	27.6	41.0	44.9	264.8
BT(95%)	28.5	41.3	49.7	428.6
BT(100%)	30.4	41.5	51.0	441.8
BT(avg)	27.7	41.0	44.6	296.0
AFR	23.8	30.6	42.7	337.9
GP	23.9	30.6	43.5	333.1
GP #trees	3	2	3	53
AFR #trees	299	299	299	100

In Table II, the optimal downloading time is computed from by (28). The results indicate that the gradient algorithm obtains near the theoretically optimal solution.

1) Comparison with BitTorrent: Table II shows the time when 50%, 95% and 100% receivers finish downloading, and the average downloading time in BitTorrent, respectively. The average downloading time is defined as the sum of the receiver downloading times divided by the number of receivers. Alternatively speaking, the BT(50%) time is the median of the downloading times, the BT(100%) time is the maximum of the downloading times and the BT(avg) time is the mean of the downloading times. The gradient algorithm outperforms BitTorrent with respect to the worst-case (maximum) downloading time. In Profile 4, where the receiver-side upload bandwidth is extremely heterogeneous, the average downloading time of BitTorrent is 37 minutes shorter than that of the gradient algorithm; however, the worst-case downloading time of BitTorrent is more than 100 minutes longer than that of the gradient algorithm. This means in the case when BitTorrent suffers from the last-chunk problem severely, the gradient projection algorithm can sacrifice the average downloading time to improve the worst-case downloading time greatly. In all other profiles, the gradient algorithm is always better than BitTorrent with respect to the average downloading time.

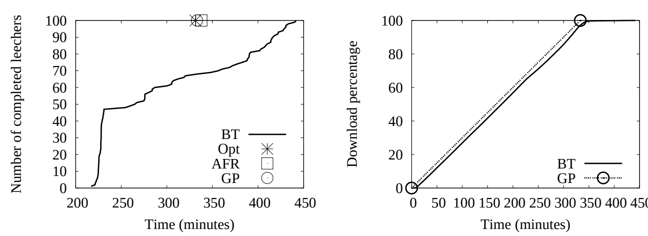
BitTorrent's performance is not bad compared with the optimal value. This was explained in [34], which models the downloading time of BitTorrent. It shows that, in the case that a flash crowd arrives at the same time, the bandwidth constraint is at the access links, and the receivers stay after they finish downloading, BitTorrent achieves near optimal distribution speed. In Fig. 3 and 4, we show the performance comparison of different distribution schemes under profile 1 and 4. The results under profile 2 and 3 are omitted for brevity. 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 two lines have different slopes, we can extrapolate the lines and expect the gradient algorithm to do much better if the file size becomes larger. This observation seems to contradict the conclusion in [34].

BitTorrent has the advantage that it does not need to maintain the routing tables and the trees explicitly. However, since we are dealing with infrastructure-based content distribution networks with low network dynamics and moderate sizes, configuring routing tables and maintaining trees in such an environment are infrequent events; they are not necessarily the difficult part in such networks, as long as the number of trees is not too large. The costs of the algorithm can be justified by the economic gains from better network utilization or performance.



(a) (b)

Fig. 3. Profile 1: (a) number of leechers that have completed download over time; (b) download percentage over time.



(a) (b)

Fig. 4. Profile 4: (a) number of leechers that have completed download over time; (b) download percentage over time.

2) Comparison with AFR: The AFR downloading time is given by (27). Table II shows that 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. The reason is that, in AFR, every receiver is required to relay all chunks it receives from the source to other receivers. This unnecessary constraint leads to a sub-optimal solution. AFT doesn't allow the breadth-first search tree, but an optimal two-phase tree does. Nevertheless, AFR achieves good performance in this access-limited situation.

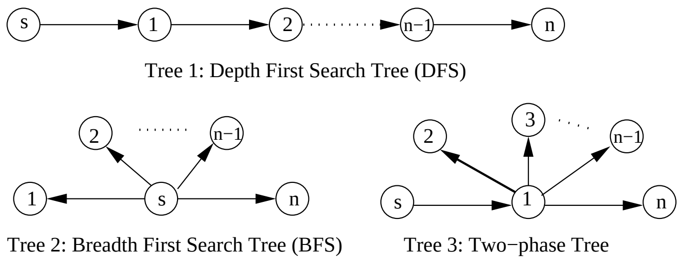


Fig. 5. Structures of trees on the overlay network

We also compare the number of active trees AFR and the gradient algorithm eventually use. The gradient algorithm uses fewer trees than AFR. We inspected the active trees. With the access-link constraint, the gradient algorithm uses three kinds of trees, the depth-first search tree (DFS), the breadth-first search tree (BFS), and a two-phase tree. Fig. 5 shows the structures of the three types of trees on the overlay network. In general, there exists an optimal solution that uses only two-phase trees for networks with such a star topology. But it seems that the gradient algorithm prefers the chain-like DFS tree. It may appear counter-intuitive that such a chain-like distribution path is preferred because the chain seems to involve largest delay. However, this is in fact not true because of our fluid model of traffic and because we do not consider propagation delay. The bit that arrives at node 1 from the source can immediately be transmitted to node 2, and to node 3, so on. We leave it to future work on how to incorporate the propagation delay in optimal tree selection.

Furthermore, transmitting chunks by a chain-like DFS tree can ensure fairness, i.e., a receiver downloads the same amount of chunks as it uploads to others. It is well known that BitTorrent incorporates a tit-for-tat (TFT) incentive mechanism to encourage contribution or prevent a node from downloading much more content than it can upload [28]. Table III shows that, when the bottleneck is at either the download links or the source (i.e., in Profiles 1 and 2), the gradient algorithm naturally prefers the DFS tree while other unfair solutions might also exist. When the aggregate upload bandwidth is the bottleneck, the algorithm tries to distribute the chunks over the two-phase trees. In addition, the more heterogenous the receivers are, the more two-phase trees we need and the more bandwidth is allocated to the two-phase trees. We can see that a two-phase tree apparently allows unfairness by inspecting its structure. In Profile 4, the receiver-side upload bandwidth is extremely heterogeneous. The optimal solution with respect to the worst-case downloading time has to give up fairness: Half of the receivers serve others at 100 Kbps and half of the receivers serve others at 1 Kbps, while all of them download at the same rate. The key conclusion here is that the gradient algorithm is able to find the best distribution trees for the particular network environment, and try to ensure fairness if at all possible. Without the help of the gradient algorithm, what types of trees are selected is not always obvious.

C. 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

TABLE III
THE DISTRIBUTION OF BANDWIDTH ALLOCATED FOR DIFFERENT TREES

	Profile 1	Profile 2	Profile 3	Profile 4
DFS	99.4%	99.999%	98.91%	1.93%
BFS	0.26%	0	0.754%	0.95%
Two-phase	0.33%	0.001%	0.333%	97.1%

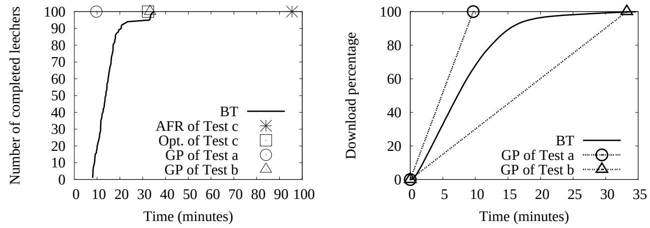
wish to see how our algorithm performs in infrastructure-mode content distribution. We did experiments with the ISP Sprintlink's backbone obtained from [27]. The network has 315 backbone nodes and 1944 links. It is the largest backbone ISP with the highest node degree among the six commercial backbone 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).

We did several experiments with the link capacities uniformly distributed in some range. The actual link capacity data is unavailable. We find the gradient algorithm often gives trivial optimal solutions. After inspecting the solutions and the network graphs, it turns out that the ISP backbone is poorly connected. There are many links that lies on all the routing paths between one peer and all other peers, which means if any one of these critical links is removed, at least one peer will be disconnected. If these critical links do not have much larger capacity than other links, they are likely to become the bottleneck. The gradient algorithm is able to locate the bottleneck immediately. Other five ISP backbones show the same property. Presumably in reality, the ISPs are aware of such links and would ensure they have very large bandwidth so that they are never the bottleneck. In order to test our algorithm in this non-trivial scenario, 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.

1) Comparison with BitTorrent: Fig. 6 shows that, in Test a, the downloading time in the gradient algorithm is only 30% of that in BitTorrent. The average downloading time of BitTorrent is 14.8 minutes, and is about 6 minutes longer than that of the gradient algorithm. 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



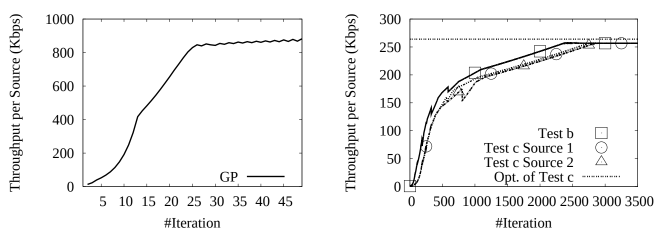
(a) (b)

Fig. 6. Profile 5. (a) Number of receivers that have completed download over time; (b) download percentage over time.

extrapolate the lines and expect the gradient algorithm to do much better if the file size becomes larger.

2) Comparison with AFR: Fig. 6 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 good solution.

3) Convergence Speed: Fig. 7 shows the convergence of the algorithm in Test a, b and c respectively. The time spent on one iteration is about one round trip time plus the time to compute the MDSP. 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, more than 2000 trees for each source. The reason that Test b has much more trees than Test a might have to do with the fact that Test b has much more constraints than Test a. In Test a, we need to satisfy the physical link capacity constraints, while in Test b, we need to satisfy the overlay link capacity constraints; there are much more overlay links than physical links. According to the discussion at the end of Section III-A, the number of links plus the number of sources serves as a rough bound of the number of active trees. It is possible that 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 direction to pursue.



(a) (b)

Fig. 7. Profile 5. Convergence of throughput. Two sources with $r_s = 1.0$ (a) Test a; (b) Test b and c.

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

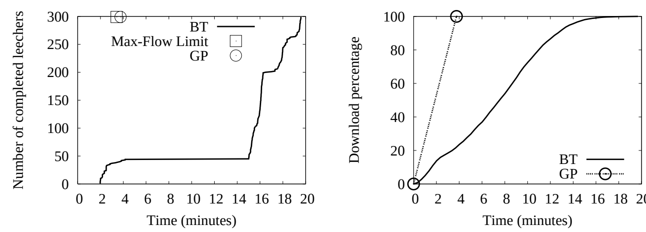
In reality, the cross-ISP links are often the bandwidth bottleneck. The goal here is to evaluate the effectiveness of our algorithm in handling the bottleneck at cross-ISP links when it is deployed at ISP gateways. We did experiments both on an artificial small cross-ISP network and the cross-ISP network 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 access 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.

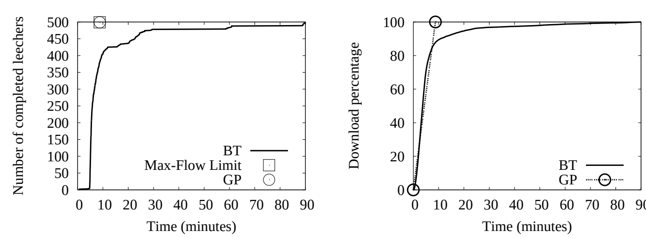
TABLE IV
DOWNLOADING TIME (MINUTES) COMPARISON OF PROFILE 6 AND 7

	BT(50%)	BT(95%)	BT(100%)	BT(avg)
P6	16	19.5	19.6	14.7
P7	5.83	26.5	90	10.9
	GP	AFR	MFL	
P6	3.8	142.7	3.4	
P7	8.74	131.2	8.72	



(a) (b)

Fig. 8. Profile 6. (a) Number of receivers that have completed download over time; (b) download percentage over time.



(a) (b)

Fig. 9. Profile 7. (a) Number of receivers that have completed download over time; (b) download percentage over time.

In both profiles 6 and 7, the gradient algorithm approaches the max-flow limit and beats BitTorrent and AFR by a large amount, up to a factor of 10 (Tabel IV: 50%, 95% and 100% are the percentage of the peers that have finished downloading. With respect to the average downloading time, the gradient algorithm is also better than BitTorrent. Also see Fig. 8 and 9.). We found, with more peers per ISP, BitTorrent's performance deteriorates. FastReplica's performance is far worse than both the gradient algorithm and BitTorrent, and we didn't even show it in Fig. 8 and 9.

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. This is because, in Profile 6, each max-flow between the source ISP and the destination ISP has the same value. This is not the case in Profile 7. Thus, the optimal solution does allow multiple copies to be sent to one destination ISP. 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.

E. 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 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. At the beginning, we have one source and 299 receivers; at iteration 100, 50 receivers leave; at iteration 200, 50 new receivers arrive; at iteration 300, 50 receivers leave (40 receivers are from the receivers that were present at the beginning, and the other 10 are from the new arrivals); at iteration 400, another 50 new receivers arrive; at iteration 500, 50 receivers leave (30 receivers are from the receivers present at the beginning, 10 receivers are from those that joined at iteration 200, and the other 10 are from the those that joined at iteration 500); at iteration 600, 50 more receivers arrive. In the interval between iteration 0 and 200, we have one session consisting of either 299 or 249 receivers. In the interval between iteration 200 and 300, we have two sessions: One consists of one original source and 299 receivers and the other consists of 250 sources with overlapping chunks and the 50 new receivers. We assume the two sessions have equal demand rates (though this might be unfair). In the interval between iteration 300 and 400, we still have two sessions but with fewer receivers. In the interval between iteration 400 and 500, we have three sessions with equal demanded rates: The first consists of one original source and 299 receivers, the second consists of 210 sources with overlapping chunks and 90 receivers, and the third consists of 250 sources with overlapping chunks and 50 receivers. In the interval between iteration 500 and 600, we have three sessions with fewer receivers. From iteration 600, we have four sessions with equal demanded rates: The first consists of one original source and 299 receivers, the second consists of 180 sources with overlapping chunks and 120 receivers, the third consists of 210 sources with overlapping chunks and 90 receivers, and the fourth consists of 250 sources and 50 receivers. Fig. 10 shows the algorithm adapts to the dynamics quickly.

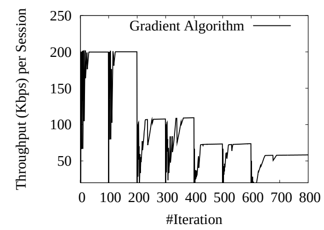


Fig. 10. Dynamic departure and arrival of receivers

VI. ADDITIONAL RELATED WORK

In our optimization-based approach, we follow the tradition of Kelly et. al. [35] and Low et. al. [36] 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., [37]–[40]. Several other related studies, either in topics or methods, are [41]–[44].

The authors of [45] formulate several optimization problems related to swarming with different objectives such as minimizing the server load, maximizing the distribution rate, or minimizing the depth of the distribution trees. Some of the formulations have the constraint that the node degree on the distribution trees must be limited. They assume that the bottleneck is the uplinks and the optimal solutions are obtained by exploiting this special condition. Their approach does not seem to be extensible to general networks where bottleneck can be anywhere. The authors of [46] consider similar optimization problems under the users' uplink capacity constraint. They present a $(1+\epsilon)$-approximation algorithm for solving various problems with different topology constraints: full or non-full mesh graph, limited or unlimited tree degree, with or without helpers. The main part of their algorithm comes from the technique for solving the maximum concurrent flow problem given in In [48], the authors study the problem of minimizing the average distribution time across the peers in an upload-constrained P2P system. The average distribution time is a useful metric to evaluate the performance of a file distribution system. However, their approach does not seem to be extensible to general networks.

VII. CONCLUSION

This paper represents a systematic study on how best to conduct content distribution using advanced swarming techniques over infrastructure networks. In response to growing 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 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 advanced 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. Wieland, "DigiWorld 2006: Voice and low prices drive FTTH in Japan," *Telecommunications Online*, Nov. 15 2006, http://www.telecommagazine.com.
[2] J. George, "FTTH design with the future in mind," *Broadband Properties*, September 2005.
[3] 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.
[4] BitTorrent Website, http://www.bittorrent.com/.
[5] L. Cherkasova and J. Lee, "Fastreplica: Efficient large file distribution within content delivery networks," in *Proceedings of the 4th USITS*, Seattle, WA, March 2003.
[6] J. Lee and G. de Veciana, "On application-level load balancing in FastReplica," *Computer Communications*, vol. 30, no. 17, pp. 3218–3231, November 2007.
[7] D. Kostić, A. Rodriguez, J. Albrecht, and A. Vahdat, "Bullet: high bandwidth data dissemination using an overlay mesh," in *Proceedings of 19th ACM Symposium on Operating Systems Principles (SOSP '03)*, October 2003.
[8] 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.
[9] B.-G. 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.
[10] 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)*, San Jose, CA, May 2006.
[11] 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.
[12] Akamai Website, http://www.akamai.com.
[13] 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.
[14] 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.
[15] M. Grotchel, A. Martin, and R. Weismantel, "Packing Steiner trees: a cutting plane algorithm and computation," *Mathematical Programming*, vol. 72, pp. 125–145, 1996.
[16] 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.
[17] D. Bertsekas and R. Gallager, *Data Networks*, 2nd ed. Prentice Hall, 1991.
[18] D. Bertsekas, *Nonlinear Programming*, 2nd ed. Athena Scientific, 1999.
[19] 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.
[20] D. Bertsekas and J. N. Tsitsiklis, *Parallel and Distributed Computation: Numerical Method*. Athena Scientific, 1997.
[21] R. E. Tarjan, "Finding optimum branchings," *Networks, Vol. 7*, pp. 25–35, 1977.
[22] L. F. P. M. Camerini and F. Maffioli, "A note on finding optimum branchings," *Networks, Vol. 9*, pp. 309–312, 1979.
[23] P. A. Humblet, "A distributed algorithm for minimum weight directed spanning trees," *IEEE Transactions on Communications*, pp. 756–762, June 1983.
[24] 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.
[25] R. G. Gallager, "A minimum delay routing algorithm using distributed computation," *IEEE Transactions on Communications*, pp. 73–85, January 1977.
[26] J. N. Tsitsiklis and D. P. Bertsekas, "Distributed asynchronous optimal routing in data networks," *IEEE Transactions On Automatic Control*, vol. AC-31, pp. 325–332, 1986.
[27] *Rocketfuel: An ISP Topology Mapping Engine*, University of Washington, http://www.cs.washington.edu/research/networking/rocketfuel/.
[28] A. R. Bharambe and C. Herley, "Analyzing and improving BitTorrent performance," Microsoft Research, Tech. Rep. MSR-TR-2005-03, 2005.
[29] 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.
[30] 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.
[31] 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.
[32] 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.
[33] J. Edmonds, "Edge-disjoint branchings," in *Combinatorical Algorithms*, R. Rustin, Ed. Academic Press, 1973, pp. 91–96.
[34] D. Qiu and R. Srikant, "Modeling and performance analysis of BitTorrent-like peer-to-peer networks," in *Sigcomm'04*, 2004.
[35] 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.
[36] 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.
[37] X. Lin and N. B. Shroff, "Joint rate control and scheduling in multihop wireless networks," in *Proceedings of the 43rd IEEE CDC*, 2004.
[38] 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.
[39] 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.
[40] 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.
[41] Y. N. 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.
[42] P. Key, L. Massoulié, and D. Towsley, "Path selection and multipath congestion control," in *Proceedings of INFOCOM 2007*, May 2007.
[43] R. Bindal, P. Cao, W. Chan, J. Medval, G. Suwala, T. Bates, and A. Zhang, "Improving traffic locality in BitTorrent via biased neighbor selection," in *Proceedings of the International Conference on Distributed Computing Systems (ICDCS'06)*, 2006.
[44] H. Zhang, G. Neglia, D. Towsley, and G. L. Presti, "On unstructured file sharing networks," in *Proceedings of INFOCOM*, May 2007.
[45] S. Liu, R. Zhang-Shen, W. Jiang, J. Rexford, and M. Chiang, "Performance bounds for peer-assisted live streaming," in *ACM SIGMETRICS*, Annapolis, MD, USA, June 2008, pp. 313–324.
[46] S. Sengupta, S. Liu, M. Chen, M. Chiang, J. Li, and P. A. Chou, "Streaming capacity in peer-to-peer networks with topology constraints," Princeton University, Tech. Rep., 2008.
[47] N. Garg and J. Köenemann, "Faster and simpler algorithms for multicommodity flow and other fractional packing problems," in *Proceedings of the 39th Annual Symposium on Foundations of Computer Science*, November 1998, pp. 300–309.
[48] G. M. Ezovski, A. Tang, and L. L. Andrew, "Minimizing average finish time in p2p networks," in *Infocom 2009*, April 2009.