The Delay Region for P2P File Transfer

Yunnan Wu*, Y. Charlie Hu†, Jin Li*, and Philip A. Chou*
*Microsoft Research, One Microsoft Way, Redmond, WA 98052. {yunnanwu, jinl, pachou}@microsoft.edu,
†Dept. of ECE, Purdue University, West Lafayette, Indiana 47907. ychu@purdue.edu

***Abstract*—Motivated by P2P file transfer applications (e.g., BitTorrent) on the Internet, this paper considers the problem of delivering a file from a server to multiple receivers in a P2P network. Each receiver has an associated delay in receiving the file. We aim at understanding the optimal delay region, i.e., the set of all possible delay vectors that can be achieved. Previous work has addressed the problem of delivering the file to all receivers in minimum amount of time (equivalently, minimizing the maximum delay to the receivers), assuming peer uplinks are the only bottleneck in the network. This paper shows that it is in fact possible to significantly reduce the average delay at a slight increase in the maximum delay. Moreover, given an order at which the receivers finish downloading, the optimal delay region is characterized by a system of linear inequalities. Any point in the optimal delay region can be achieved by linear network coding. We also propose a simple routing scheme that has near-optimal empirical performance.**

I. Introduction

Peer-to-peer file transfer is a very popular class of applications in today's Internet. In a P2P file download application, the key performance metric from an end-user's point of view is the delay, or the time it takes to download the file. Practical P2P file transfer solutions (e.g., BitTorrent) try to speed up the downloading process by efficiently leveraging the peer uplink bandwidths. These solutions generally improve the delays perceived by the users. However, the fundamental performance limit (in terms of achievable delays) remains unclear. In this paper, we are interested in characterizing the optimal delay region, i.e., the set of all achievable delay vectors.[1]

Consider a source node s that wants to broadcast a file of size B to a set of N receivers, $\{1, \ldots, N\}$, in a peer-to-peer network. We start with a popular though simplified model for the peer-to-peer network. It is assumed that peer uplinks are the only bottlenecks in the whole network[2], and every peer can connect to every other peer through routing in the overlay. Let c_v denote the uplink capacity constraint for node v. Let T_i denote the time for the i-th receiver to receive the file; collectively, let $\boldsymbol{T}$ denote an achievable delay vector.

[1]To focus on the fundamental limit from a network information theory point of view, we assume that all nodes are cooperative, unlike the BitTorrent protocol, which uses a Tit-For-Tat mechanism to ensure nodes have incentive to contribute their upload resources. The cooperative assumption also holds naturally in many practical scenarios, e.g., in "closed" content distribution systems where the programs are managed by a single authority.

[2]This assumption can be partly justified by the empirical observation that in the overwhelming majority of residential broadband connections, bottlenecks typically are at the edge of the access networks rather than in the middle of the Internet. Furthermore, for typical residential connections (e.g., DSL and Cable), the uplink capacity is often several times smaller than the downlink capacity.

Previous studies [3]–[5] have determined that assuming a full-mesh topology and assuming peer uplinks are the only bottleneck, the minimum amount of time to finish all nodes is:

$$\min\left\{\frac{B}{c_s},\ \frac{BN}{c_s + \sum_{i=1}^{N} c_i}\right\}. \quad (1)$$

The fact that the time to finish all nodes cannot be lower than (1) follows from simple resource counting arguments. The first term in (1) is the time it takes for the source to send one copy of the file out; the second term is the total amount of work (BN bits) divided by the aggregate upload resource. Conversely, it was shown in [3] that (1) can indeed be achieved by communicating using a number of spanning trees, using a construction known in [3] as MutualCast.

The MutualCast construction in fact lets all the receivers finish at the same time, equal to (1). Clearly, (1) is the minimum possible value of $\max_{i=1}^{N} T_i$. Is it possible for some receivers to finish earlier such that $1/N \sum_{i=1}^{N} T_i < \max_{i=1}^{N} T_i$?

For any receiver i, $T_i \geq B/c_s$ because the source has to be able to send one copy of the file out. If the bound (1) is dominated by the first term, i.e., if the source's uplink is the bottleneck, then the best we can hope for is to finish every node by time B/c_s, which can be achieved, for example, by the MutualCast scheme. Hence if $B/c_s \leq BN/(c_s + \sum_{i=1}^{N} c_i)$, the optimal delay region is given by $\{\boldsymbol{T} | T_i \geq B/c_s,\ \forall i\}$. In the rest of this paper, we focus on the remaining case, where

$$c_s \leq \frac{\sum_{i=1}^{N} c_i}{N-1}. \quad (2)$$

Consider an example where a source s is distributing a file of size $B = 1$ to three receivers $1, 2, 3$. Suppose $c_s = 2.5$ and $c_1 = c_2 = c_3 = 1.0$. With the MutualCast scheme, all receivers will finish at time $6/11 = 0.55$. Now consider an alternative scheme, given by Figure 1. First, in time interval $[0, 0.4]$, we use three multicast trees to transfer the file to nodes 1 and 2. The three MutualCast trees are shown in solid lines, dotted lines, and dashed lines, respectively; the amount of data flowing in each edge is labelled on the edge. At the end of this time interval, nodes 1 and 2 have finished; node 3 has received 0.2 unit of information. Next, in time interval $[0.4, 26/45]$, nodes $s, 1, 2$ collaboratively upload to node 3. Thus $T_1 = T_2 = 0.4$ and $T_3 = 26/45 = 0.58$ and $\sum_i T_i = 1.38$.

In comparison, the average delay of the solution in Figure 1 is 16% shorter than the MutualCast solution and the maximum delay of this solution is only 5% longer. Thus this example shows that it is possible to reduce the average delay by letting

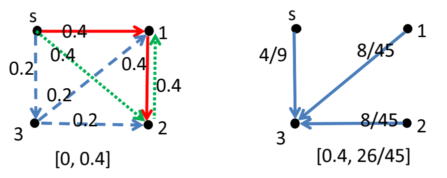


Fig. 1. A constructive scheme for a 4-node example where $B = 1$, $c_s = 2.5$, and $c_1 = c_2 = c_3 = 1$.

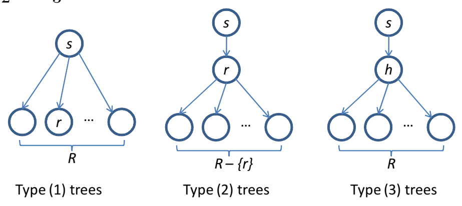


Fig. 2. The different types of MutualCast trees.

some nodes finish early. More generally, the objective of this paper is to characterize the entire set of achievable delay vectors and design low-complexity constructive schemes.

II. A Simple Bound

To establish the simple bound presented in this section, we make use of a version of the MutualCast capacity theorem [3] that allows helper nodes, which are nodes other than the source and the receivers that can help.

Lemma 1 (MutualCast with helpers [3]): Consider a full-mesh P2P topology in which peer uplinks are the only bottleneck. Let c_v be the uplink capacity constraint for node v. Consider a single multicast session given by source node s, a set of receivers R, and a set of helper nodes H. Then the maximum achievable broadcast rate is:

$$\min\left\{ c_s,\ \frac{c_s + \sum_{v\in R} c_v + \frac{|R|-1}{|R|}\sum_{h\in H} c_h}{|R|} \right\}. \tag{3}$$

This maximum achievable broadcast rate can be achieved by packing at most $1 + |R| + |H|$ trees as follows:

- One depth-1 tree rooted at s and reaching all receivers in R, i.e. the type (1) tree in Figure 2.
- $|R|$ depth-2 tress, each rooted at s and reaching all other receivers in R via different $r \in R$, i.e. the type (2) tree in Figure 2.
- $|H|$ depth-2 trees, each rooted at s and reaching all receivers in R via different $h \in H$, i.e., the type (3) tree in Figure 2.

Lemma 2: Consider a P2P topology in which peer uplinks are the only bottleneck. Assume that the nodes are labeled in decreasing order of their uplink capacities, i.e., $c_1 \geq c_2 \geq \ldots \geq c_N$. For any scheme, let $T_{[1]} \leq T_{[2]}, \ldots, \leq T_{[N]}$ denote the sorted sequence of the download times of the N receivers. Then the following must hold:

$$T_{[i]} \geq \frac{B}{\min\{c_s,\ D_i\}} \tag{4}$$

where

$$D_i \triangleq \frac{c_s + \sum_{j=1}^{i} c_j + \frac{i-1}{i}\sum_{j=i+1}^{N} c_j}{i}. \tag{5}$$

This implies a lower bound on the sum delay,

$$\sum_{i=1}^{N} T_i = \sum_{i=1}^{N} T_{[i]} \geq \sum_{i=1}^{N} \frac{B}{\min\{c_s,\ D_i\}} \tag{6}$$

Proof: Note that by time $T_{[i]}$, i nodes have finished. Let R denote these i nodes and let H denote the other $N - i$ nodes, who can act as helpers. Then applying Lemma 1, we see that

$$T_{[i]} \geq \frac{B}{\min\left\{ c_s,\ \frac{c_s + \sum_{v\in R} c_v + \frac{|R|-1}{|R|}\sum_{h\in H} c_h}{|R|} \right\}}.$$

Note that the right hand side of the above inequality is minimized when R is the set of nodes with the i largest uplinks. Hence $T_{[i]} \geq B/\min\{c_s,\ D_i\}$. ■

One natural question to ask is whether the N inequalities in (4) can all be achieved with equality simultaneously. This, in general, is not the case. To see this, consider the earlier example where a source s is distributing a file of size $B = 1$ to three receivers $1, 2, 3$, and $c_s = 2.5$, $c_1 = c_2 = c_3 = 1.0$. From Lemma 2, we obtain the following bound on the sum delay: $\sum_i T_i = \sum_i T_{[i]} \geq \frac{1}{2.5} + \frac{1}{2.5} + \frac{6}{11} = 1.35$. Suppose there is a scheme that achieves this bound with equality. Then the scheme must finish node 1 at time $1/2.5 = 0.4$, node 2 at time $1/2.5 = 0.4$, and node 3 at time $6/11$. Since $B/D_2 = \frac{2}{c_s + c_1 + c_2 + 1/2c_3} = 0.4$, all upload resource must be fully utilized. In particular, node 3, as a helper, must receive $0.4/2 = 0.2$ and send 0.4 in time $[0, 0.4]$. Now by time 0.4, nodes 1 and 2 have finished and node 3 must have received 0.2 unit of information. The earliest time we can finish node 3 is thus $0.4 + 0.8/(2.5+1+1) = 0.58 > 6/11 = 0.55$. In general, the bound is not achievable because although MutualCast can provide a scheme to achieve the minimum possible $T_{[i]}$ for each i, there is no single scheme that can achieve the minimum possible $T_{[i]}$ for all i.

III. The Optimal Delay Region

The bound in Lemma 2 is constructed by asking N separate questions without forcing the solutions to be consistent with each other over time. As a result, in general the N bounds in (4) cannot be simultaneously achieved. To obtain a complete characterization of the achievable delay region, we need to somehow force consistency among the solutions over time. In this section we show how to achieve this by introducing states along the time axis and considering a time-expanded graph. The general idea of using time-expansion to handle causality is known; for example, it was used in [1] to prove a capacity result for network coding in cyclic graphs. For our context, the time-expanded graph has a different structure and it is used for a different purpose – investigating the achievable delay vectors.

First, we introduce some notions. We divide the time into N *epochs* accordingly to the finishing times of the nodes. Thus one node finishes at the end of the first epoch; a second node finishes at the end of the second epoch; and so on. Let

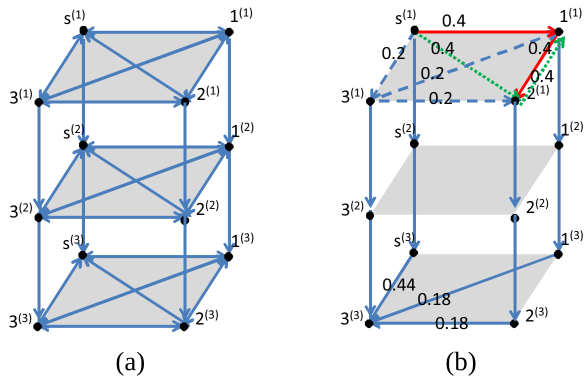


Fig. 3. (a) The expanded graph for the 4-node example. (b) The expanded graph with a traffic assignment, for the constructive solution in Figure 1.

Δt_i denote the length of the i-th epoch. This provides an alternative parameterization because i receivers finish by time $T_{[i]} = \sum_{j=1}^{i} \Delta t_i$.

Given a P2P graph G with node set $V = \{s, 1, \ldots, N\}$ and allowed link set E, we introduce a time-expanded graph $G^{(N)}$ as follows. For each $v \in V$ and $n \in \{1, \ldots, N\}$, $G^{(N)}$ includes a vertex $v^{(n)}$; this vertex corresponds to the associated physical node in the n-th epoch. The graph $G^{(N)}$ has two types of edges.

1. For each $e \in E$ going from u to v and each $n \in \{1, \ldots, N\}$, the graph $G^{(N)}$ also includes an edge $e^{(n)}$ going from $u^{(n)}$ to $v^{(n)}$; such an edge corresponds to the transmission from u to v during the n-th epoch.
2. For each $v \in V$ and each $n \in \{1, \ldots, N-1\}$, the graph $G^{(N)}$ also includes an edge with infinite capacity from $v^{(n)}$ to $v^{(n+1)}$. These edges represent the accumulation of received information by node v over time. They are called *memory edges*.

For the 4-node example we have been using, the corresponding time-expanded graph $G^{(N)}$ is illustrated by Figure 3(a). This graph consists of 3 layers, corresponding to the 3 epochs, with forward memory connections along the time line. Thus the i-th layer characterizes the information flows in the i-th epoch; when we move from one epoch to the next, the information a node learned in the past is available to it in the next epoch. The beauty of this graph is that it explicitly models the operations in different epochs, in a single graph. For example, the constructive scheme in Figure 1 can be represented by Figure 3(b). More generally, any feasible scheme corresponds to a traffic assignment in $G^{(N)}$, where each edge in $G^{(N)}$ is labelled with the number of bits the edge carries.

We next show how to search for good solutions via linear programming. Before doing that, we shall review the notion of *flow*. Consider a graph $G = (V, E)$ where the edges have capacity constraints. The capacity of edge $e \in E$ is $c(e)$; for our purpose, think of the capacity of an edge as the maximum number of bits that can be sent over the edge. In G, given a source node $s \in V$ and a destination node $t \in V$, an s–t *flow* is a nonnegative vector $\boldsymbol{f}$ of length $|E|$ satisfying the *flow conservation constraint*: $\text{excess}_v(\boldsymbol{f}) = 0, \quad \forall v \in V - \{s, t\}$, where

$$\text{excess}_v(\boldsymbol{f}) \equiv \sum_{e \in In(v)} f_e - \sum_{e \in Out(v)} f_e. \tag{7}$$

Let $\mathcal{F}_{s,t}(B)$ denote the set of s–t flows, each with its flow value equal to B. Then $\boldsymbol{f} \in \mathcal{F}_{s,t}(B)$ if and only if

$$\begin{aligned} &\boldsymbol{f} \geq \boldsymbol{0}, \\ &\text{excess}_s(\boldsymbol{f}) = -B, \\ &\text{excess}_v(\boldsymbol{f}) = 0, \quad \forall v \in V - \{s, t\}. \end{aligned}$$

From the Max-Flow-Min-Cut Theorem, in a traffic assignment from a feasible scheme (e.g., Figure 3(b) from the scheme in Section IV), if node i recovers the file at the end of epoch κ_i, then there must exist an $s^{(1)}$–$i^{(\kappa_i)}$ flow with value B. The converse direction in fact also holds. This follows from network coding theory.

Lemma 3 (Network Coding for Multicasting [1], [2]): Consider a directed graph $G = (V, E)$ with edge capacities specified as a length-$|E|$ vector $\boldsymbol{c}$. Consider a multicast session where a source s wants to multicast information to a set of receivers T at rate r. The traffic demand can be fulfilled in $(V, E, \boldsymbol{c})$ if and only if there exists a set of flows $\{\boldsymbol{f}_t\}$ such that:

$$\boldsymbol{c} \geq \max_{t \in \boldsymbol{T}} \boldsymbol{f}_t, \tag{8}$$

where $\boldsymbol{f}_t \in \mathcal{F}_{s,t}(r)$ is an s–t flow with rate r in $(V, E, \boldsymbol{c})$, for all $t \in T$. Furthermore, if (8) holds, then there exists a linear network coding solution.

Due to the structure of the time-expanded graph $G^{(N)}$, our original problem boils down to one of finding a multicast solution in the time-expanded graph $G^{(N)}$, for any given order of the nodes to be finished. Given an order at which the nodes will be finished, say node i finishes at epoch κ_i, the corresponding multicast session has source node $s^{(1)}$ and receiver set $\{i^{(\kappa_i)}\}$. Then applying network coding theory, we obtain a characterization of the feasible set of download times, or equivalently the feasible set of epoch durations $\{\Delta t_i\}$. This is stated in the following theorem.

Theorem 1 (All Feasible Download Times): Consider a P2P network in which peer uplinks are the only bottleneck. Consider multicasting information from a source node s to a set of receiver nodes $\{1, \ldots, N\}$. Given an order at which the nodes will be finished, say node i finishes at epoch κ_i, a set of epoch durations $\{\Delta t_i\}$ are feasible if and only if the following system of linear inequalities has a feasible solution:

$$\boldsymbol{g} \geq \boldsymbol{f}_i, \quad i = 1, \ldots, N, \tag{9}$$

$$\boldsymbol{f}_i \in \mathcal{F}_{s^{(1)}, i^{(\kappa_i)}}(B), \quad i = 1, \ldots, N, \tag{10}$$

$$\sum_{w:\ v^{(i)} w^{(i)}} g_{v^{(i)} w^{(i)}} \leq c_v \Delta t_i, \quad \forall v \in V, \forall i, \tag{11}$$

$$\Delta t_i \geq 0, \quad \forall i. \tag{12}$$

Here the variables are $\boldsymbol{g}, \boldsymbol{f}_i, \Delta t_i$.

Theorem 1 can be extended in a number of ways. For instance, it can be extended to optimize any linear objective of the variables (e.g., the average delay) and cover other types of network constraints (e.g., downlink constraints). Note that the above linear system of inequalities has a description that is linear in the problem size. Hence for a fixed ordering of the receivers, the optimal solution can be found in polynomial time. The involvement of the download ordering, however, appears to be combinatorial. For the P2P network model that we have been considering, where there are only uplink constraints, we conjecture that the optimal ordering is to finish the nodes from the largest to the smallest uplink capacity (because the bound in Section II is tightest with such ordering). Note that in the special case where all receiver nodes have the same uplink capacity, the ordering does not matter due to symmetry. Hence in such symmetric setup, the optimal solution (for any linear objective) can be found in polynomial time.

Corollary 1 (Polynomial Time): Consider a P2P network in which peer uplinks are the only bottleneck and all peers have the same uplink capacity. The minimum average delay (in fact, any linear objective in Δt_i) can be found in polynomial time.

IV. A Routing-Based Scheme

Theorem 1 provides a theoretical characterization of the delay region. However, to use it in practice, we need to solve a linear program (to search for a good operating point and decide the flow assignment) and the solution may require network coding. The resulting complexity may still be considered too high for a large network. In this section, we propose a simple routing scheme that can achieve good performance. Algorithm 1 gives one constructive scheme, which is a generalization of the solution given in Figure 1.

Algorithm 1 A Constructive Scheme

```
1: Let $k$ be the smallest index such that $c_s \geq D_k$.
2: In the first epoch, use MutualCast trees to deliver the file
   to nodes $1, \ldots, k$ in minimum possible time, $B/D_k$.
3: for $i = k+1, \ldots, N$ do
4:     In the $i$-th epoch (i.e., $[T_{i-1}, T_i]$), all nodes except node
       $i$ upload to node $i$, and node $i$ uploads to node $i+1$.
       If $i = N$, then node $N$ does not upload.
5: end for
```

We assume $c_1 \geq c_2 \geq \ldots \geq c_N$. In line 1 of Algorithm 1, we let k be the smallest index such that $c_s \geq D_k$. Note that $k > 1$ because $D_1 = c_s + c_1 > c_s$. Furthermore, it can be verified that $D_j \geq D_{j+1}$ for $j > 1$. Thus $c_s \geq D_k > D_{k+1} \ldots > D_N$. For the example given in Figure 1, $c_s = 2.5$, $D_1 = 3.5$, $D_2 = 2.5$, and $D_3 = 11/6$; hence $k = 2$.

The algorithm uses MutualCast trees to deliver the file to k nodes in minimum possible time, $B/\min\{c_s, D_k\} = B/D_k$. In this case, since the second term in the MutualCast capacity expression is smaller, all resources are fully utilized and the solution is unique. More specifically, for $i = 1, \ldots, k$, each receiver node i uses a type (2) tree (Figure 2) to send out $c_i \Delta_1$ bits, exhausting all its upload resource. For $j = k+1, \ldots, N$, each helper node j uses a type (3) tree (Figure 2) to send out a total of $c_j \Delta_1$ bits, exhausting all its upload resource. Then the source uses its remaining bandwidth to distribute additional content using a type (1) tree. As a result, at the end of the first epoch, nodes $1, \ldots, k$ have received the entire file, and each node j for $j \in \{k+1, \ldots, N\}$ has received $c_j \Delta t_1/k$ bits. Furthermore, the bits that nodes $k+1, \ldots, N$ have received are distinct. For the example in Figure 1, the three MutualCast trees are shown in solid lines, dotted lines, and dashed lines, respectively; the amount of data flowing in each edge is labelled on the edge. At the end of the first epoch, nodes 1 and 2 have finished; node 3 have received 0.2 unit of information.

Next, in the $(k+1)$-th epoch, all nodes except node $k+1$ upload to node $k+1$, and node $k+1$ uploads to node $k+2$. Thus node $k+1$ downloads in parallel at rate $c_s + \sum_{i=1}^{N} c_i - c_{k+1}$. Since node $k+1$ receives $c_{k+1}\Delta t_1/k$ bits in the first epoch, it only needs $B - c_{k+1}\Delta t_1/k$. Thus if these uploaders have enough bits to serve node $k+1$, then it can finish after time

$$\Delta t_{k+1} = \frac{B - \frac{\Delta t_1 c_{k+1}}{k}}{c_s + \sum_{i=1}^{N} c_i - c_{k+1}}. \tag{13}$$

For the example in Figure 1, in the third epoch, node 3 downloads in parallel from nodes $s, 1, 2$. Since node 3 needs to obtain 0.8 unit of information, the third epoch last for $\Delta t_3 = \frac{1-0.2}{2.5+1+1} = 0.18$.

However, for (13) to hold, we have to check to ensure that indeed all nodes $k+1, \ldots, N$ have enough bits to upload to node $k+1$. To establish that we need to show that $\Delta t_{k+1} c_j \leq c_j \Delta t_1/k$, where the right hand side is the number of bits that node j for $j \in \{k+1, \ldots, N\}$ received in the first epoch. This is established next.

Claim 1: $\Delta t_{k+1} \leq \Delta t_1/k$.

Proof:

$$\Delta t_{k+1} \leq \Delta t_1/k \tag{14}$$

$$\iff \frac{B - \frac{\Delta t_1 c_{k+1}}{k}}{c_s + \sum_{i=1}^{N} c_i - c_{k+1}} \leq \Delta t_1/k \tag{15}$$

$$\iff B \leq (c_s + \sum_{i=1}^{N} c_i)\Delta t_1/k \tag{16}$$

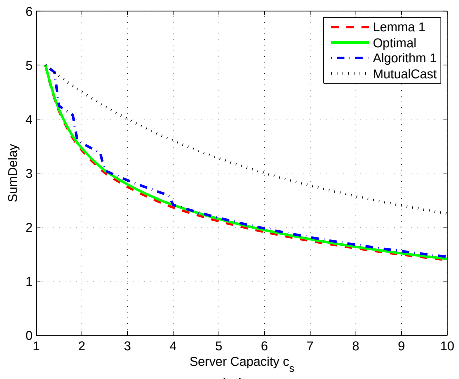
$$\iff \frac{Bk}{c_s + \sum_{i=1}^{N} c_i} \leq \Delta t_1 \tag{17}$$

On the other hand, we know that

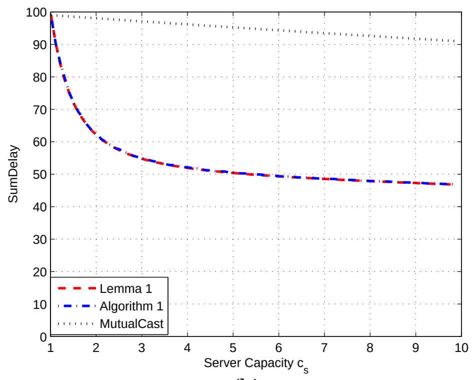
$$\Delta t_1 = \frac{B}{D_k} = \frac{Bk}{c_s + \sum_{i=1}^{k} c_i + \frac{k-1}{k}\sum_{i=k+1}^{N} c_i}.$$

Hence the claim. ■

During the $(k+1)$-th epoch, node $k+1$ uses its upload bandwidth to upload to node $k+2$. It can be verified that $\Delta t_{k+1} c_{k+1} \leq \Delta t_1 c_{k+1}/k \leq B - \Delta t_1 c_{k+2}/k$. Thus node $k+2$



(a)



(b)

Fig. 4. (a) Evaluation of the results for $N = 6$, $B = 1$, $c_1 = \ldots = c_N = 1$. (b) Evaluation of the results for $N = 100$, $B = 1$, $c_1 = \ldots = c_N = 1$.

is not finished at the end of the $(k+1)$-th epoch and it has $\frac{\Delta t_1 c_{k+2}}{k} - \Delta t_{k+1} c_{k+1}$ bits in stock.

Proceeding similarly (by showing $\Delta t_j \leq \Delta t_1/k$, for $j > k$, etc), it can be shown that the scheme is indeed feasible and its download times satisfy:

$$\Delta t_1 = \frac{B}{D_k}, \tag{18}$$

$$\Delta t_2 = \ldots = \Delta t_k = 0, \tag{19}$$

$$\Delta t_{k+1} = \frac{B - \frac{\Delta t_1 c_{k+1}}{k}}{c_s + \sum_{i=1}^{N} c_i - c_{k+1}}, \tag{20}$$

$$\Delta t_j = \frac{B - \frac{\Delta t_1 c_j}{k} - \Delta t_{j-1} c_{j-1}}{c_s + \sum_{i=1}^{N} c_i - c_j}, \quad \text{for } j = k+2, \ldots, N. \tag{21}$$

V. Evaluation

In this section, we evaluate the bounds and the schemes in the paper. Figure 4(a) presents the results for $N = 6$ and $c_1 = \ldots = c_6 = 1$. We vary the server capacity c_s from $\sum_i c_i/(N-1)$ to 10 and collect the sum delay. There are four curves, corresponding to: the bound given by Lemma 2, the optimal result given by Theorem 1, the simple routing-based scheme given by Algorithm 1, and the MutualCast scheme. It is seen that the bound given by Lemma 2, the optimal, and Algorithm 1 are all very close. For the setup in Figure 1, where $c_s = 2.5$, the solution given in Section IV seems to be optimal (the sum delay of Algorithm 1 is 62/45=1.3778; the linear program for the optimal result returns 1.3778).

Figure 4(b) presents the results for $N = 100$ and $c_1 = \ldots = c_N = 1$. The bound given by Lemma 2 and the simple routing-based scheme given by Algorithm 1 are hardly distinguishable from each other. For $c_s = 10$, the sum delay achieved by Algorithm 1 is 46.85, which is 51.5% that of the MutualCast scheme ($N * N/(N-1) = 90.91$).

Note that if a scheme can achieve a sum delay very close to the lower bound (6), then $T_{[i]}$ of the scheme must be close to optimal, for each i. Such a scheme would lead to a desirable operating point in the delay region because it is essentially "universally" near-optimal. Thus Algorithm 1 is seen to be a low-complexity algorithm with good performance.

VI. Conclusion

In this paper we formulated the problem of finding the set of all achievable delays for information broadcasting in a P2P network. We presented three results.

Section II provides a closed-form bound obtained by bounding the delays separately for the node that finishes the first, then the node that finishes the second, and so on. Each individual delay bound follows from existing theoretical results on multicasting in P2P networks. The bound is generally loose, because it does not force consistency over time.

Section III shows how to use the technique of time-expansion to introduce states along the time axis and thus force a consistent solution over different epochs. This results in a necessary and sufficient characterization of the delay region. Furthermore, the technique is a general technique that can be used to characterize delay tradeoff problems in other, non-P2P, networks.

Section IV presents a simple routing-based scheme, built directly on the intuition that to reduce the delays, it is better to concentrate the resources and process the N "jobs" sequentially. For a single server serving N equal-length jobs, sequential processing of jobs can reduce the average delay to half, compared to parallel processing of jobs. The network information multicast problem, however, is more complicated because of information causality issues. We address such challenge by using a first stage of content distribution (to ensure every node gets some data to serve others) followed by sequential, concentrated serving. The end result is a simple and near-optimal algorithm that can reduce the average delay up to half compared to state-of-the-art P2P file transfer solutions.

References

[1] R. Ahlswede, N. Cai, S.-Y. R. Li, and R. W. Yeung. Network information flow. *IEEE Trans. Inform. Theory*, 46(4):1204–1216, July 2000.

[2] S.-Y. R. Li, R. W. Yeung, and N. Cai, Linear network coding, *IEEE Trans. Inform. Theory*, 49(2):371–381, Feb. 2003.

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

[4] D. M. Chiu, R. W. Yeung, J. Q. Huang, and B. Fan, Can network coding help in P2P networks. In *2nd Workshop on Network Coding (NetCod)*. Apr. 2006.

[5] R. Kumar, Y. Liu, and K. W. Ross, Stochastic fluid theory for P2P streaming systems. In *Infocom 2007*, Anchorage, Alaska. IEEE, 2007.