Improving File Distribution Performance by Grouping in Peer-to-Peer Networks

Ma Lingjun, Pui-Sze Tsang, and King-Shan Lui

***Abstract*—It has been shown that the peer-to-peer paradigm is more efficient than the traditional client-server model for file sharing among a large number of users. Given a group of leechers who wants to download a single file and a group of seeds who possesses the whole file, the minimum time needed for distributing the file to all users can be calculated based on their bandwidth availabilities. A scheduling algorithm has been developed so that every leecher can obtain the file within this minimum time. Unfortunately, this mechanism is not optimal with regard to the *average download time* among the peers. In this paper, we study how to reduce the *average download time* without prolonging the time needed for all leechers to obtain the file from a theoretical perspective. Based on the bandwidth capacities, the seeds and leechers are divided into different groups. We identify the necessary conditions for grouping to bring about benefits. We also study the impact on performance when leechers leave the system before the downloading process is complete. To evaluate our mechanism, we conduct extensive simulations and compare the performance with a BitTorrent-like file sharing algorithm. The results show that our grouping protocol successfully reduces the *average download time* over a wide range of system configurations.**

***Index Terms*—Peer-to-peer, grouping, file distribution.**

I. Introduction

FILE sharing under the peer-to-peer (P2P) model is more efficient than the traditional client-server model, and this is demonstrated by applications such as BitTorrent (BT) [1] and PPLive [2]. In the P2P paradigm, each user acts both as a client and a server, sending file pieces that he/she possesses to other users while receiving pieces he/she needs from others. This enables P2P file sharing to scale well when the number of participating nodes increases [3]. In this paper, we study the performance of distributing a single file offline to a large number of users who do not have any portion of the file in the beginning. Several servers, called *seeds*, possess the whole file before the distribution begins. The users who want to obtain the file are called the *leechers*. A leecher can connect to any seed and any leecher for sharing the file. The rate at which a seed/leecher can upload/download is constrained by its bandwidth capacity. One important performance metric is the time needed to distribute the file to all leechers.

File scheduling is a common technique used to reduce the file distribution time [4] [5] [6] [7]. A scheduling scheme determines who should send a file piece to whom and when the file piece should be sent out. Two approaches, pull-based and push-based, have been developed. In the pull-based approach, a peer selects a piece it does not possess and requests it from other peers. On the other hand, in the push-based situation, a peer selects a piece it possesses and sends it to other peers. The work in [8] shows that a purely push-based or pull-based mechanism is not efficient in file distribution. A hybrid approach called INTERLEAVE is proposed, which combines the two approaches to reduce the file distribution time. Unfortunately, an optimal scheduling mechanism is difficult to develop, especially in a large network. Network coding is proposed to solve this problem [9] [10]. Instead of sending a certain file block, a linear combination of the blocks available in the sender is sent. Although experimental results show that the distribution time is better than some commonly used heuristic block distribution mechanisms, the information distributed is more than the information needed in a perfect scheduling mechanism.

Based on the fluid model, Kumar and Ross derived the theoretical lower bound of the file distribution time [4]. They also developed an optimal file distribution scheme that achieves this lower bound. In other words, if their algorithm can distribute the file to *all* leechers in t seconds, there is no other scheduling algorithm that is able to finish the task in less that t seconds. In their scheme, all leechers complete their downloads at the same time, which means that the *average download time* of the leechers is also t seconds. However, the time needed may not be the best from an individual leecher's perspective. Note that the time needed for a *single* leecher is related to its download capacity. The smaller the capacity, the longer time it takes. If there are some leechers with extremely limited download capacities, as all leechers finish at the same time in the algorithm of Kumar and Ross, other leechers have to wait even if their capacities are abundant. For this reason, we believe that the *average download time* can be further reduced by grouping seeds and leechers appropriately. Peers can only exchange file pieces with other peers in the same group, and inter-group communication is not allowed. As each peer can only talk to a subset of the whole system, the overhead in maintaining inter-peer communication is reduced as well.

In this paper, we study how the average download time can be reduced by clustering from a theoretical perspective. We first identify the necessary condition for grouping to be feasible in bringing about benefits. We then develop a determinstic grouping scheme that is promising in reducing the *average download time* in file distribution. The scheme can

Manuscript received July 31, 2008; revised May 4, 2009. The associate editor coordinating the review of this paper and approving it for publication was J.P. Martin-Flatin.

The authors are with the Department of Electrical and Electronic Engineering, the University of Hong Kong, Hong Kong (e-mail: kslui@eee.hku.hk).

The project was funded in part by the University of Hong Kong Seed Funding Programme for Basic Research, Project No. 200710159006.

Digital Object Identifier 10.1109/TNSM.2009.03.090302

work with different scheduling algorithms such as BT and the algorithm developed by Kumar and Ross [4]. Moreover, it is compatible with network coding. Finally, we extend the model in [4] and analyze the system performance in a dynamic model where some leechers fail before the file distribution process is completed. In summary, following the theoretical studies conducted by Kumar and Ross [4], we study the impact of grouping in P2P file sharing. Our results offer insights into practical protocol design where grouping can be adopted as a strategy for performance improvement.

The rest of this paper is organized as follows. Section II presents the related work, and Section III defines the problem studied in this paper. Our grouping mechanism is discussed in Section IV. Section V studies how the failure of a leecher affects the system. The criteria for the system performance to experience degradation is also described. Performance evaluation with regard to the grouping protocol is given in Section VI. We conclude the whole paper in Section VII.

II. Related Work

Previous studies on P2P file distribution are mainly based on two models: the chunk model and the fluid model. In the chunk model, files are first chopped into pieces of equal sizes, and the subsequent distribution occurs in pieces. That is, a peer will not distribute a piece until he/she has received the last bit of that piece. As opposed to the chunk model, it is assumed in the fluid model that a peer can distribute a bit once that bit is received.

One important evaluation metric in file distribution is the minimum amount of time needed to distribute the file to all leechers. We refer to this value as the *minimum download time*. In the chunk model, it is difficult to develop a closed form expression for the minimum download time. By assuming all peers have the same upload capacities and unlimited download bandwidth, Mundinger *et al.* developed a closed form expression for the minimum download time[1] [5]. Other researchers analyzed the minimum download time based on certain scheduling mechanisms. The work in [11] studied the performances of several tree-based distributing schemes. The works in [8] assumed file distribution is slot-based, where each user can upload one piece at most in each slot, and there is no limit on a user's download capability. A centralized optimal protocol is developed that can complete the file distribution in $k + log_2 n$ slots, where there are k file pieces, n leechers, and a single source. The popular chunk-based BitTorrent (or BT-like) protocol was also studied [3] [12].

There are some analytical frameworks developed based on the fluid models. In [13] [14], a stochastic fluid model was built to study the performance of the P2P web cache (SQUIRREL) and cache clusters. The request streams of the individual nodes are modeled by a fluid flow, and the large number of users can join or leave the system randomly. The work in [15] applied the Fluid Stochastic Petri Net (FSPN) to analyze the distribution of the transfer time in P2P file sharing systems. The authors analyzed the transfer time distribution in file sharing applications and further extended their model by including features such as parallel downloads and on-off peer behavior [16]. However, none of the frameworks mentioned above attempted to derive the *minimum download time* for the file distribution process. Kumar and Ross [4] derived the first closed form expression of the minimum download time based on the fluid model and designed a scheduling algorithm to achieve it. In their model, every leecher can talk to every seed/leecher, and each node has limited upload and download capacities. However, peers in [4] are assumed to stay in the system until the whole distribution process ends, making the *average download time* to be the same as the minimum download time. This is not very advantageous because a peer cannot finish earlier even he/she has abundant bandwidth. In this paper, we study how grouping can reduce the average download time without increasing the minimum download time.

Grouping has been used to solve problems of content location and discovery in peer-to-peer networks. Peers with similar interests are aggregated into groups, and this significantly improves the content search capabilities of the P2P network as a whole. The work in [17] proposes a multi-group structure, which is self-organized and based on the interests of peers in the system. A locality-based clustering peer-to-peer overlay network (LCO) is introduced in [18]. LCO restricts each flood to the range of one cluster. Thus, it reduces the unnecessary traffic produced by the topology mismatching between the P2P logical overlay network and the physical underlying network. In [19], an interest-based clustering peer-to-peer network (ICN), which applies Freenet [20] mechanism and cache management, is introduced. The works in [21] employ the *scale-free* and cluster properties of user interests in query searching to enhance efficiency. As peers tend to group with peers of similar interest, an interest-based proximity measurement scheme is described in [22] for query searching. While these schemes are based on peers' interests, our grouping scheme is based on the peers' bandwidth information, which is effective in reducing the *average download time*.

Although BT does not explicitly group peers, clustering of peers with similar bandwidth can be observed in its operation as well [23]. The tracker protocol could be extended to report peer capacities so that peers could select peers of similar bandwidth. However, it was not clear how to determine whether two bandwidth values were similar. Furthermore, the performance was not studied from a theoretical perspective. When the bandwidth of communication between two nodes also depends on the path between them, it would be beneficial to establish sessions of higher bandwidth for file sharing. As peers that are closer in terms of network distance tend to have better bandwidth capacities, the Ono plugin for BT [24] identifies these peers based on the network measurements. Our work, on the other hand, assumes the bandwidth bottlenecks are on the seeds and leechers as in [4] [25] [26].

We study the problem of offline file download. Another related issue that also attracts some attention is P2P multicast for multimedia streaming. Traditionally, an application layer multicast is used to send the whole stream from the source node to the receivers. In a P2P multicast system, the stream is divided into different pieces, and each piece follows a different

[1]In their paper, *minimum download time* is referred to as minimal distribution time.

multicast tree [25] [27] [26]. As a multimedia stream is shared, it is assumed that not all the file pieces are necessary, which is very different from the requirement of offline file download. SplitStream [25] is the first system that uses a *forest* (multiple trees) for stream sharing. The trees built are *interior-node-disjoint*, meaning that if a peer is an internal node on a certain multicast tree of a file piece, it will be a leaf node in all other multicast trees used for other file pieces. ChunkySpread [27] allows a peer to be an internal node of more than one tree. It also considers *latency threholds* in the tree construction to reduce the latencies of the stream distribution. Orchard [26] avoids free-riders in the multicast by ensuring that a peer makes a fair contribution. All these systems focus on forest construction that allows peers to join and leave dynamically. The related performance metrics include whether a new peer can successfully join a tree, workload of a peer, latency, number of file pieces obtained, and others. In this work, on the other hand, we are interested in the *average download time* of the file downloading process in which every peer has to receive every bit of a file, and no peer joins or leaves in the middle of the distribution process. We adopt the KR algorithm [4] as the scheduling mechanism, which has been formally proven to be the optimal scheduling algorithm for offline file downloading based on the fluid model.

III. Preliminaries

A. Notations and Definitions

In this section, we define the notations used in this paper, and the summary can be found in Table I.

TABLE I
Table of Notations

Symbol	Meaning
S	set of seeds
L	set of leechers
$u(x)$	upload bandwidth of seed/leecher x
$d(x)$	download bandwidth of leecher x
$u(S)$	total upload bandwidth of all seeds in S, $\sum_{s\in S} u(s)$
$u(L)$	total upload bandwidth of all leechers in L, $\sum_{l\in L} u(l)$
d^L_{min}	minimum download bandwidth among the leechers in L, $\min\{d(l)\|l \in L\}$
$T_{min}(S,L,F)$	minimum possible download time for the seeds in S to distribute a file of size F to the leechers in L (F may be omitted for simplicity)
$T^A_{avg}(S,L,F)$	average download time needed for the seeds in S to distribute a file of size F to the leechers in L using Algorithm A
$T^G(S,L)$	time needed for the leechers in L to get the file from seeds in S when grouping is applied

We denote the set of seeds as S, and the set of leechers as L. Following the model in [4], each leecher can obtain file pieces from every seed and every leecher. All seeds and leechers join the distribution process at the same time, and they stay in the system until the distribution is complete. Each seed is associated with an upload bandwidth limiting how much data it can send out at a time. The upload bandwidth is independent of which leecher the seed is sending to. That is, the limit of sending to Leecher A alone is the same as the limit of sending to Leecher B alone. There is also no limit on how many leechers the seed is sending the file pieces to. It can use all its bandwidth to send to only one leecher or split the bandwidth among several leechers. Similar to the seeds, each leecher also has its own upload bandwidth. They need to download as well; thus, they are also associated with download bandwidth, which tells how much at most each leecher can receive at a time.

The upload and download bandwidth of a seed/leecher x are denoted as $u(x)$ and $d(x)$, respectively. The minimum download bandwidth of the leechers in a leecher set L is d^L_{min}. That is, $d^L_{min} = min\{d(l)|l \in L\}$. We further define $u(S)$ to be the total upload bandwidth of all seeds in S and $u(L)$ to be the total upload bandwidth of all leechers in L. Formally, $u(S) = \sum_{s\in S} u(s)$ and $u(L) = \sum_{l\in L} u(l)$. For the example seed set (S) and leecher set (L) in Tables II and III, we have d^L_{min} = 500 Kbps, $u(S)$ = 18300 Kbps, and $u(L)$ = 10900 Kbps.

The goal of file distribution is to allow all leechers, which do not have any single file piece in the beginning, to obtain the whole file. A scheduling algorithm governs how the seeds and leechers work together to distribute the file. Every leecher takes a certain amount of time to obtain the whole file. We denote the *download time of a single leecher* l as $T(l)$. When a group of leechers are considered, different leechers may have different download times. The *download time of a group of leechers* L is the time it takes for *all* leechers to obtain the file. Therefore, it corresponds to the time needed for the last leecher to finish downloading the file, which is $max\{T(l)|l \in L\}$. We denote $T(S,L,F)$ to be the *download time of the leechers in* L *given the seed set* S *for downloading a file of size* F.

Different scheduling algorithms yield different $T(S,L,F)$. Kuman and Ross analyzed the minimum possible $T(S,L,F)$ that can be achieved based on the fluid model where peers are assumed to stay for as long as the file distribution takes [4]. We refer this time as the *minimum download time* and denote it as $T_{min}(S,L,F)$. Eq. (1) gives the formula of $T_{min}(S,L,F)$.

$$\begin{aligned} & max\{\frac{F}{d^L_{min}}, \frac{|L|F}{u(L)+u(S)}, \frac{F}{u(S)}\} \\ = \quad & \frac{F}{min\{d^L_{min}, \frac{u(S)+u(L)}{|L|}, u(S)\}} \end{aligned} \tag{1}$$

$\frac{F}{u(S)}$ represents the amount of time needed for the seeds to send out a single copy of F. $\frac{F}{d^L_{min}}$ is the amount of time needed for the leecher(s) with the smallest bandwidth (d^L_{min}) to obtain a copy of F. Finally, $\frac{|L|F}{u(L)+u(S)}$ reflects the time needed for both seeds and leechers to contribute all their upload bandwidth to the distribution of $|L|$ copies of F to all leechers. As a complete file distribution should perform all the three tasks, $T_{min}(S,L,F)$ is the maximum value among

the three terms.

Note that Eq. (1) is independent of the scheduling algorithm. That is, *no* algorithm can distribute the file to all leechers using time less than $T_{min}(S, L, F)$. An optimal scheduling algorithm is an algorithm that can distribute the file using exactly $T_{min}(S, L, F)$ amount of time. Kumar and Ross also developed such an optimal algorithm, and we refer this algorithm as the *KR Algorithm* in the rest of this paper. This algorithm allows *every leecher* to obtain the whole file in $T_{min}(S, L, F)$ amount of time. That is, every leecher finishes downloading at the same time. If we define the *average download time* of leechers in L as $\frac{\sum_{l \in L} T(l)}{|L|}$, the average download time of the KR algorithm is the same as $T_{min}(S, L, F)$. We denote the average download time of the KR algorithm as $T^{KR}_{avg}(S, L, F)$. That is, $T^{KR}_{avg}(S, L, F) = T_{min}(S, L, F)$. [We write $T_{min}(S, L, F)$ and $T_{avg}(S, L, F)$ as $T_{min}(S, L)$ and $T_{avg}(S, L)$ when the context is clear. In the rest of the paper, F will be omitted in the notations for simplicity.]

TABLE II
UPLOAD BANDWIDTH OF SEEDS

Seed	s_1	s_2	s_3	s_4	s_5
$u(s_i)$/Kbps	300	1200	4800	5600	6400

TABLE III
UPLOAD AND DOWNLOAD BANDWIDTH OF LEECHERS

Leecher	l_1	l_2	l_3	l_4	l_5
$u(l_i)$/Kbps	1800	1500	1450	1400	1300
$d(l_i)$/Kbps	1000	500	2000	9500	7000
Leecher	l_6	l_7	l_8	l_9	l_{10}
$u(l_i)$/Kbps	850	800	650	600	550
$d(l_i)$/Kbps	6500	9600	7400	4500	1300

Refer to the seeds and leechers in Tables I and II. Assume F = 300M. If we apply the KR algorithm, $T_{min}(S, L) = T^{KR}_{avg}(S, L)$ = 600 s, which means every leecher in Table III takes 600 s to obtain F. However, it is definitely not the best for individual leechers. For example, if Seed 3 sends the whole file to Leecher 3, Leecher 3 takes only 150 s to receive the file. Of course, if we do so, other leechers may take more time to finish. Is it possible to reduce the download time of individual leechers without prolonging the time for the last leecher to finish? That is, is it possible to reduce the *average download time* while maintaining the download time to be $T_{min}(S, L)$? We found that if we divide the seeds and the leechers into different groups and a node can only communicate with the nodes in the same group, it is possible to reduce the *average download time*. To illustrate, we divide our example seed set and leecher set as in Fig. 1. We denote the seeds as s_i, where $1 \leq i \leq |S|$ and the leechers as l_i, where $1 \leq i \leq |L|$. One group consists of seed set $S_3 = \{s_4\}$ and leecher set $L_3 = \{l_5, l_7, l_9\}$. If we apply the KR algorithm, $T(S_3, L_3)$ = 108.4 s. The *minimum download times* of other groups are shown in the figure. It can be observed that $max\{T(l)|l \in L\}$ under this grouping arrangement is still 600 s. However, the *average download time* of the leechers is only 227.7 s. Therefore, grouping can further improve the performance of the KR algorithm, and we aim at developing an effective grouping mechanism.

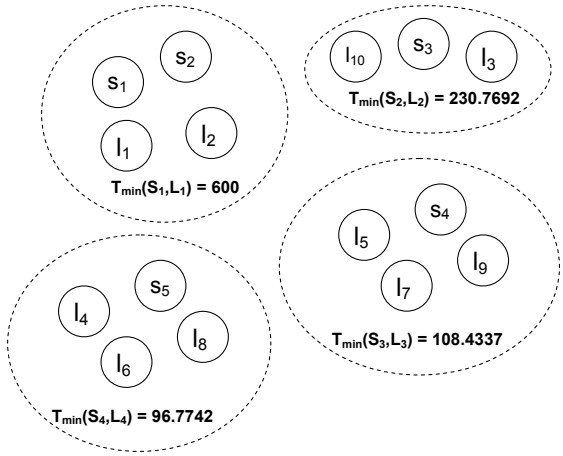

Fig. 1. Grouping the seeds and the leechers in Tables I and II.

We now formally define a grouping. Given a set of seeds S and a set of leechers L, a grouping scheme G divides the seeds and the leechers into $K (K \geq 2)$ groups, G_1, G_2, ..., G_K, where S_i and L_i belong to G_i, and the following conditions are satisfied:

1) The subsets of S are S_1, S_2, ..., S_K, where $S_i \cap S_j = \emptyset$ if $i \neq j$ and $\cup_{1 \leq i \leq K} S_i = S$.
2) The subsets of L are L_1, L_2, ..., L_K, where $L_i \cap L_j = \emptyset$ if $i \neq j$ and $\cup_{1 \leq i \leq K} L_i = L$.

Leechers in L_i can only obtain file pieces from other leechers in L_i and the seeds in S_i, and no inter-group communication is allowed. Therefore, if the KR algorithm is adopted in each group, $T(l) = T_{min}(S_i, L_i)$ for $l \in L_i$. Let $T^G_{avg}(S, L)$ be the *average download time* after grouping G is implemented, and the KR algorithm is adopted in each group. Then,

$$T^G_{avg}(S, L) = \frac{\sum_{i=1}^{K} |L_i| T_{min}(S_i, L_i)}{|L|}.$$

We are interested in a grouping that achieves smaller *average download time* without increasing the *minimum download time*. We refer a grouping with this feature as *feasible grouping*. That is, the feasible grouping G of K groups satisfies the following:

1) $T_{min}(S_i, L_i) \leq T_{min}(S, L)$, where $1 \leq i \leq K$
2) $T^G_{avg}(S, L) < T_{min}(S, L)$

To evaluate the benefit of feasible grouping G, we define the performance improvement ratio ρ as follows:

$$\rho = \frac{T^{KR}_{avg}(S, L)}{T^G_{avg}(S, L)} = \frac{|L| T_{min}(S, L)}{\sum_{i=1}^{K} |L_i| T_{min}(S_i, L_i)} \quad (2)$$

For the seeds and the leechers in Tables I and II, after applying the grouping scheme illustrated in Fig. 1, the performance improvement ratio $\rho = \frac{600s}{227.7s} = 2.635$, which indicates a significant performance improvement.

B. Existence of Feasible Grouping

Although grouping looks promising in reducing the *average download time*, we find that a feasible group may not always exist. In this section, we identify the conditions that imply the impossibilities of finding a feasible grouping. If there exists a feasible grouping for the configuration (S, L), we say that (S, L) is *feasible in grouping*.

Denote $T^G(S, L)$ as the download time of grouping G with K groups. Then,

$$T^G(S, L) = max\{T_{min}(S_i, L_i)|i = 1, 2, \ldots, K\}. \quad (3)$$

We study the existence of feasible grouping in three cases, with one case for each term in Eq. (1).

Case A: $T_{min}(S, L) = |L|F/(u(L) + u(S))$
Case B: $T_{min}(S, L) = F/u(S)$
Case C: $T_{min}(S, L) = F/d^L_{min}$

Lemma 1: No feasible grouping exists in Case A.
Proof:
In Case A, for any grouping scheme G that divides the seeds and the leechers into $K \geq 2$ groups,

$$\begin{aligned} T_{min}(S, L) &= |L|F/(u(L) + u(S)) \\ &= \frac{\sum_{i=1}^{K} |L_i|F}{\sum_{i=1}^{K} (u(S_i) + u(L_i))} \\ &\leq max\left\{\frac{|L_i|F}{u(S_i) + u(L_i)}|i = 1, 2, \ldots, K\right\} \\ &\leq max\{T_{min}(S_i, L_i)|i = 1, 2, \ldots, K\} \\ by\ (3) \quad &= T^G(S, L). \end{aligned}$$

The download time after grouping should be no larger than that before grouping. Therefore, we need $T_{min}(S, L) = T^G(S, L)$, which holds true if and only if

$$\frac{|L_1|}{u(S_1) + u(L_1)} = \cdots = \frac{|L_K|}{u(S_K) + u(L_K)} = \frac{|L|}{u(S) + u(L)}$$

This implies that $T^G_{avg}(S, L) \geq T_{min}(S, L)$, and no feasible grouping exists. □

Lemma 2: No feasible grouping exists in Case B.
Proof:
In Case B, $T_{min} = F/u(S)$. As $F/u(S_i) > F/u(S)$ for every proper subset S_i of S, we have $T^G(S, L) > T_{min}(S, L)$ for any grouping G. Therefore, there is no feasible grouping in this case. □

We have studied the situations in Cases A and B. We will study Case C in the next section.

C. NP-Completeness of Grouping

It turns out that some configurations satisfying the condition of Case C are feasible in grouping while some are not. In other words, $T_{min}(S, L) = F/d^L_{min}$ is not a sufficient condition to determine the existence and non-existence of feasible grouping. For example, the configuration in Tables I and II satisfies Case C. The grouping as illustrated in Fig. 1 is a feasible grouping. On the other hand, if there are only two seeds s_1 and s_2, where $u(s_1) = 300$ Kbps and $u(s_2) = 18000$ Kbps, to distribute the file to the leechers in Table II, no feasible grouping exists. In fact, it is NP-complete to find a feasible grouping. To prove NP-completeness, we first develop the following lemma.

Lemma 3: If G' is a feasible grouping of configuration (S, L), and G' consists of more than two groups, then there exists a feasible grouping G that consists of exactly two groups.

Proof:
We prove the lemma by showing how to construct G based on the K groups of G'. As G' is feasible,

$$\frac{F}{d^{L_i}_{min}} \leq \frac{F}{d^L_{min}} \text{and} \frac{F}{u(S_i)} \leq \frac{F}{d^L_{min}} \text{and} \frac{|L_i|F}{u(S_i) + u(L_i)} \leq \frac{F}{d^L_{min}}$$

for $i = 1, 2, \ldots, K$. Furthermore, there is at least one group with a distribution time less than $T_{min}(S, L)$. Without loss of generality, we assume (S_K, L_K) is the group that follows $T_{min}(S_K, L_K) < T_{min}(S, L)$. Let $S_a = S_1 \cup S_2 \cup \cdots \cup S_{K-1}$, $L_a = L_1 \cup L_2 \cup \cdots \cup L_{K-1}$, $S_b = S_K$, and $L_b = L_K$. That is, $u(S_a) = u(S_1) + u(S_2) + ... + u(S_{K-1})$ and $d^{L_a}_{min} = min\{d^{L_i}_{min}|i = 1, 2, \ldots, K-1\}$. We have

$$\begin{aligned} \frac{F}{d^{L_a}_{min}} &= \frac{F}{min\{d^{L_i}_{min}|i = 1, 2, \ldots, K-1\}} \\ &\leq \frac{F}{d^L_{min}}, \\ \frac{F}{u(S_a)} &= \frac{F}{u(S_1) + u(S_2) + \cdots + u(S_{K-1})} \\ &< \frac{F}{d^L_{min}}, \\ \frac{|L_a|F}{u(S_a) + u(L_a)} &= \frac{\sum_{i=1}^{K-1} |L_i|F}{\sum_{i=1}^{K-1} (u(S_i) + u(L_i))} \\ &\leq max\{\frac{|L_i|F}{u(S_i) + u(L_i)}|1 \leq i \leq K-1\} \\ &\leq \frac{F}{d^L_{min}}. \end{aligned}$$

Let (S_a, L_a) and (S_b, L_b) be the two groups in G. We have $T_{min}(S_a, L_a) \leq T_{min}(S, L)$, $T_{min}(S_b, L_b) < T_{min}(S, L)$, and $T^G_{avg}(S, L) < T_{min}(S, L)$. G is a feasible grouping, thus proving the lemma. □

Lemma 3 indicates that in deciding whether a Case C configuration is feasible in grouping, we only need to determine whether there exists a feasible grouping scheme that divides the seeds and the leechers into two groups. A feasible grouping scheme should maintain the download time to $T_{min}(S, L)$. That is, for individual group $G_i = S_i \cup L_i (i = 1, 2)$, $max\{\frac{F}{d^{L_i}_{min}}, \frac{|L_i|F}{u(L_i)+u(S_i)}, \frac{F}{u(S_i)}\} \leq \frac{F}{d^L_{min}}$. As $\frac{F}{d^{L_i}_{min}} \leq \frac{F}{d^L_{min}}$, we need to find a grouping such that both

$$u(S_i) \geq d^L_{min} \quad (4)$$

and

$$u(S_i) + u(L_i) \geq |L_i|d^L_{min} \quad (5)$$

are satisfied. We deal with (4) first. According to (4), for a grouping scheme to be feasible, it needs to divide the seeds into two groups such that the combined upload bandwidth of

the seeds in each group is greater than d^L_{min}. We formally define the problem as follows:

Problem 1 Given a set S of positive real numbers and $d > 0$, does a partition $\mathcal{P}$ that divides S into two subsets S_1 and S_2, where $\sum_{x \in S_i} x \geq d$ for $i = 1$ and 2, exists?

We show that it is NP-complete to decide whether there exists a feasible grouping scheme that meets (4). That is,

Lemma 4: Problem 1 is NP-complete.

For Problem 1 to be NP-complete, it needs to meet two conditions. First, Problem 1 needs to be NP; second, a well-known NP-complete problem is polynomial-time reducible to Problem 1.

Proof:

First, we prove that the problem is NP. Given a partition that divides the set into two subsets S_1 and S_2, to determine the feasibility of the scheme, we only need to obtain the sum of the elements in each subset and compare it with d. As it takes polynomial time to check the feasibility of a partition, it is NP to decide whether there exists a feasible partition.

We now prove the problem is NP-complete. Given a set A in the well-known set partition problem [28], we construct a problem instance of Problem 1 by assigning S as A and d as $\frac{1}{2}(\sum_{x \in A} x)$. The transformation is polynomial. Set A can be partitioned into two sets, B and $A \setminus B$, where the sums of the elements are the same if and only if S can be divided to two subsets S_1 and S_2, where the sum of the elements of each set is no less than $\frac{1}{2}(\sum_{x \in A} x)$. The set partition problem is polynomial-time, reducible to Problem 1. Problem 1 is therefore NP-complete. □

As it is NP-complete to decide whether we can divide the seeds in a way that the necessary condition Eq. (4) is satisfied, it is NP-complete to decide the feasibility of grouping in Case C.

IV. Grouping Protocol

The results of the previous section indicate that we can only identify a feasible grouping of a configuration (S, L) where $T_{min}(S, L) = F/d^L_{min}$. In this section, we present a mechanism to identify a feasible grouping. Formally, given S and L, we would like to find G with K groups such that

1) The subsets of S are S_1, S_2, ..., S_K, where $S_i \cap S_j = \emptyset$ if $i \neq j$ and $\cup_{1 \leq i \leq K} S_i = S$.
2) The subsets of L are L_1, L_2, ..., L_K, where $L_i \cap L_j = \emptyset$ if $i \neq j$ and $\cup_{1 \leq i \leq K} L_i = L$.
3) $T_{min}(S_i, L_i) \leq T_{min}(S, L)\ \forall\ i$
4) $T^G_{avg}(S, L) = \frac{\sum_{i=1}^{K} |L_i| \cdot T_{min}(S_i, L_i)}{|L|} < T_{min}(S, L)$

A good protocol should produce feasible grouping schemes against a wide range of networks of varying bandwidth characteristics. Due to the intricacies involved, our protocol is divided into two phases: seed grouping and leecher grouping. Given S and L, the whole grouping and verification procedure is as follows:

1) Check whether $T_{min}(S, L) = \frac{F}{d^L_{min}}$. If not, no feasible grouping exists; otherwise, proceed to 2).
2) Apply seed grouping to S.
3) Apply leecher grouping to L.

A. Seed Grouping

Section III proves that it is NP-complete to decide whether there exists a feasible seed grouping. We employ a heuristic algorithm to perform seed grouping. It is our intention to divide the seeds into as many groups as possible such that the complexity of each group is reduced.

Algorithm 1 shows the pseudocodes of seed grouping. We start by comparing $u(S)$ and d^L_{min}. If $u(S) < 2d^L_{min}$, no feasible grouping scheme exists. If $u(S) \geq 2d^L_{min}$, we first sort the seeds in non-descending order of their upload bandwidth. We then put each seed s_i into S_i. If all the seed bandwidth values are larger than d^L_{min}, we are done. Otherwise, we merge the two groups with the smallest seed upload bandwidth. The process is repeated until either all groups satisfy the condition, or all the groups are merged back to S. In the latter case, our algorithm cannot find a feasible grouping scheme.

Algorithm 1 Seed Grouping Algorithm

input: upload bandwidth of $|S|$ seeds, the minimum download bandwidth of the leechers d^L_{min};
output: grouping scheme for seeds;

```
u ← u(s_1) + u(s_2) + ... + u(s_|S|)
if u < 2d^L_min then
    output no feasible grouping scheme exists
else
    sort the seeds in S, such that u(s_1) ≤ u(s_2) ≤ ... ≤ u(s_|S|);
    Place |S| seeds into S_1, S_2, ..., S_|S|, such that s_i is in S_i;
    K ← |S|;
    while d^L_min > u(S_1) and K > 1 do
        merge S_1 and S_2 to form S_1';
        Re-sort S_1', S_3, ..., S_{K-1}, S_K in the ascending order of their upload bandwidth;
        Denote the re-sorted groups by S_1, S_2, ..., S_{K-1};
        K ← K − 1;
    end while
    if K = 1 then
        seed grouping scheme not available
    else
        output grouping scheme for seeds
    end if
end if
```

We now illustrate the seed grouping mechanism using the example in Table I. Initially five seeds, $s_1, s_2, \ldots, s_5$, are put into five groups, $S_1, S_2, \ldots, S_5$, respectively. From Table II, we know that $d^L_{min} = 500$ Kbps. We start from S_1, which is the group with the minimum upload bandwidth. As $u(S_1) < d^L_{min}$, we merge S_1 and S_2 to form S_1'. Re-sort $S_1', S_3, \ldots, S_5$ in ascending order of their upload bandwidth. Denote the re-sorted groups by $S_1, S_2, \ldots, S_4$. Then, the upload bandwidth of S_1 is 1500 Kbps. $u(S_1) > d^L_{min}$ indicates that there is no need to do further merging. Algorithm 1 divides $s_1, s_2, \ldots, s_5$ into four groups, such that $S_1 = \{s_1, s_2\}, S_2 = \{s_3\}, S_3 = \{s_4\}$, and $S_4 = \{s_5\}$. This constitutes the seed grouping scheme for seeds in Table 1.

The total time complexity of Algorithm 1 is $O(|S| log |S|)$ if heap sort is employed. For this reason, Algorithm 1 ensures the system scales well when there are a large number of seeds present. Unfortunately, due to the NP-complete nature of the problem, our polynomial time algorithm may not be able to identify a feasible grouping for every configuration (S, L). For example, let S contain six seeds, where $u(s_1) = 1$ Kbps, $u(s_2) = 1$ Kbps, $u(s_3) = 2$ Kbps, $u(s_4) = 2$ Kbps, $u(s_5) = 3$ Kbps, $u(s_6) = 3$ Kbps and $d^L_{min} = 6$ Kbps. We first put the seeds with smallest bandwidth in the same group. Then, there

are five groups with upload bandwidth units 2, 2, 2, 3, and 3, respectively. We merge the two groups with 2 units of upload bandwidth. Four groups remain, and their bandwidth units are 2, 3, 3, and 4, respectively. If we continue the merging process, all the seeds are put in the same group, and there is no partition at all. Nevertheless, there *is* a feasible one for S, which is $S_1 = \{s_1, s_3, s_5\}$ and $S_2 = \{s_2, s_4, s_6\}$.

B. Leecher Grouping

Assume K groups are derived from the seed grouping. We now describe how to assign leechers into these K groups. Algorithm 2 shows the pseudo-code of the leecher grouping. We first sort the leechers in L such that $u(l_1) \geq u(l_2) \geq \ldots \geq u(l_N)$. Starting from l_1, we assign leechers one by one into different groups. When we consider a leecher l_i, we calculate $T_{min}(S_j, L_j \cup \{l_i\})$ for $j = 1, 2, \ldots, K$, which is the *minimum download time* of G_j if l_i is assigned to G_j. Denote by Min the minimum $T_{min}(S_j, L_j \cup \{l_i\})$ among all groups. If $Min \leq \frac{F}{d^L_{min}}$ and G_c is the corresponding group that results in Min, we assign l_i to G_c. Otherwise, we merge the two groups with the least total seed upload bandwidth and re-calculate $Min \leq T_{min}(S_j, L_j \cup \{l_i\})$. The process is repeated until either $Min \leq \frac{F}{d^L_{min}}$ or all the groups are merged back to one group. In the latter case, our algorithm cannot find a feasible grouping scheme.

Algorithm 2 Greedy Leecher Grouping Algorithm

input:upload bandwidth of *N* leechers, which follows $u(l_1) \geq u(l_2) \geq \ldots \geq u(l_N)$;
output:grouping scheme for leechers

```
for i=1 To N do
    while K > 1 do
        for j = 1 To K do
            Calculate T_min(S_j, L_j ∪ {l_i});
            /*the minimum download time of G_j if l_i is assigned into G_j*/
        end for
        Min ← T_min(S, L);/*the minimum download time before grouping*/
        g ← 0; /*mark group number*/
        for j = 1 To K do
            if Min > T_min(S_j, L_j ∪ {l_i}) then
                Min ← T_min(S_j, L_j ∪ {l_i});
                g ← j;
            end if
        end for
        if Min = T_min(S, L) then
            /*leecher i cannot be assigned into any of these K groups*/
            Merge G_1 and G_2 to form G_1';
            Re-sort G_1', G_3, ..., G_K by the ascending order of their combined
            seed upload bandwidth;
            K ← K − 1;
        else
            assign leecher i into Group g;
        end if
    end while
    if K = 1 then
        grouping scheme not available;
    end if
end for
output grouping scheme for leechers
```

To illustrate, we again take the seeds and the leechers in Tables I and II as an example. After the seed grouping, $S_1 = \{s_1, s_2\}$, $S_2 = \{s_3\}$, $S_3 = \{s_4\}$, and $S_4 = \{s_5\}$. We start leecher allocation from l_1, with $u(l_1)$ = 1800 Kbps and $d(l_1)$ = 1000 Kbps. For $j = 1, 2, \ldots, 4$, we obtain $T_{min}(S_j, L_j \cup \{l_1\})$ from Eq.(1), which is the *minimum download time* of G_j if l_1 were placed in G_j. As $T_{min}(S_1, L_1 \cup \{l_1\}) = T_{min}(S_2, L_2 \cup \{l_1\}) = \cdots = T_{min}(S_4, L_4 \cup \{l_1\}) = 300$ s, l_1 is placed in the group with the smallest group number, which is G_1. We go on to consider l_2, and it is placed in G_1 as well. If l_3 were placed in G_1, current $T_{min}(S_1, L_1 \cup \{l_3\})$ would be 600 s. On the other hand, if l_3 were placed in any group other than G_1, the *average download time* would be 150 s. Therefore, l_3 is placed in G_2. The same procedure is carried out for l_4 to l_{10}, and the following is the grouping result: $G_1 = \{s_1, s_2, l_1, l_2\}$, $G_2 = \{s_3, l_3, l_{10}\}$, $G_3 = \{s_4, l_5, l_7, l_9\}$ and $G_4 = \{s_5, l_4, l_6, l_8\}$. After grouping,

$$\begin{aligned} T^G_{avg}(S, L) &= \tfrac{1}{10}(600 * 2 + 230.8 * 2 + 108.4 * 3 + 96.8 * 3) \\ &= 227.7(s), \end{aligned}$$

which indicates a significant improvement over $T_{min}(S, L)$. As K groups are formed after seed grouping, the time complexity of the greedy leecher grouping is $O(NlogN) + O(KN)$ if the heap sort is used.

In the Greedy Leecher Grouping Algorithm, leechers are considered according to the non-ascending order of their upload bandwidth. This order is critical because it ensures that the working configuration is feasible in grouping every time we consider the placement of a new leecher. To prove this, consider the placement of leecher l_i. We assume that all leechers before l_i are successfully placed in one of the groups, and these leechers are represented as L', where $1 \leq |L'| < |L|$. Under this assumption, it is obvious that grouping is feasible for seed set S and leecher set L'. Thus, it must follow that

$$\frac{u(S) + u(L')}{|L'|} > d^L_{min}.$$

We now examine whether the configuration is still feasible in grouping after we bring in l_i. Assume to the contrary that the grouping is not feasible and

$$\frac{u(S) + u(L') + u(l_i)}{|L'| + 1} < d^L_{min}. \tag{6}$$

From (6), $u(l_i) < d^L_{min}$. Consider the leecher l_{i+1} next to l_i for placement. As $u(l_{i+1}) \leq u(l_i) < d^L_{min}$, we have

$$\frac{u(S) + u(L') + u(l_i) + u(l_{i+1})}{|L'| + 1 + 1} < d^L_{min}.$$

If we continue this process until the last leecher in line for placement, we will come to the conclusion that

$$\frac{u(S) + u(L)}{|L|} < d^L_{min}. \tag{7}$$

This is a contradiction because configuration (S, L) is feasible in grouping. Therefore, if we sort the leechers in the descending order of their upload bandwidth, every time we bring in a new leecher l_i for placement, it is always true that

$$\frac{u(S) + u(L') + u(l_i)}{|L'| + 1} \geq d^L_{min}. \tag{8}$$

From (8), we find that grouping is justified in every step of the leecher grouping process.

C. Discussions

We follow the mathematical model developed in [4], and the work assumes that a group of seeds will distribute a file to a group of leechers that does not have any file piece. No seed or leecher joins or leaves before the distribution ends. Example scenarios include scheduled patch updates in which users would expect to receive a large file at around the same time. As a static environment is assumed, the grouping can be computed in a centralized manner. A seed can be elected as the coordinator, and when a leecher wants to join the file download process, it should contact the coordinator seed and report its upload and download capacities. After the coordinator seed collects all the information, it can determine the grouping and inform the seeds and leechers to form groups by themselves. After the grouping is formed, the peers in a group have to determine how they distribute the file among them. The message overhead for file distribution depends on the scheduling algorithm adopted. If the KR scheduling scheme, which will be described in the next section, is used, the schedule can be determined at the same time with the grouping algorithm. No more message is needed as long as the grouping does not change. Therefore, the message overhead involved in grouping is similar to that in the BT protocol in which a leecher needs to contact the tracker to obtain peer information.

Although we describe our protocol based on the KR algorithm, as Eq. (1) does not depend on the scheduling algorithm, our protocol can be used to divide a large configuration into smaller groups such that some groups can have a smaller theoretically best possible download time of download. In fact, each group can adopt any scheduling mechanism the peers agree on. However, not every algorithm can achieve a lower bound as specified in Eq. (1). If the theoretical download time analysis of a scheduling algorithm is available, our grouping mechanism can be enhanced to develop a grouping mechanism for this scheduling algorithm.

V. Impact of Leecher Failure

A leecher may leave the system before the file distribution ends. In this section, we address this problem and study the impact of leecher failure on the minimum download time. We demonstrate that grouping allows the isolation of the adverse effect of leecher failure. We will first give a brief overview of the KR scheduling mechanism before analyzing the performance.

A. The KR Scheduling Algorithm

We follow the notations developed in Section III-A. As there is no grouping involved in the KR algorithm, we aggregate all the seeds to a single one as S. Then, $u(S)$ represents the upload capacity of this seed. In general, the file is divided into $|L|$ chunks F_1, F_2, ..., $F_{|L|}$, and the seed distributes chunk F_i to leecher l_i directly. Leecher l_i is responsible for sending F_i to other leechers in the system (in most cases). The size of F_i and the rate of sending F_i are calculated according to which term determines the minimum download time in Eq.(1). Four different cases are considered:

Case I: $T_{min}(S,L) = F/d^L_{min}$ and $d^L_{min} \leq \frac{u(L)}{|L|-1}$

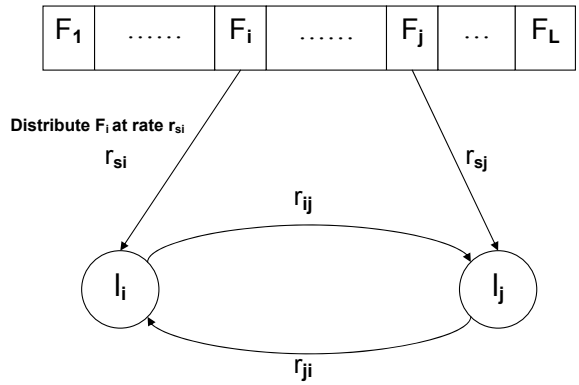

Fig. 2. The KR Scheduling Algorithm

Case II: $T_{min}(S,L) = F/d^L_{min}$ and $d^L_{min} > \frac{u(L)}{|L|-1}$
Case III: $T_{min}(S,L) = |L|F/(u(L)+u(S))$
Case IV: $T_{min}(S,L) = F/u(S)$

Figure 2 illustrates the file distribution process, and Table IV gives the rates at which the peers exchange the file chunks in different cases. The rate at which the seed sends the file to leecher l_i is r_{si}. The rate at which leecher l_i distributes information to leecher l_j is r_{ij}. Note that a leecher can distribute a file piece only after the seed has sent the piece to it. As the fluid model is assumed, $r_{si} \geq r_{ij}$ for every pair of leechers l_i and l_j. Furthermore, the total rate of l_i sends to other leechers should not exceed its limit. Therefore, $\sum_{j=1,i\neq j}^{|L|} r_{ij} \leq u(l_i)$. Similarly, the rate that l_i receives from the seed and other leechers should not be larger than $d(l_i)$. In Case I, F_i has a size $\frac{u(l_i)}{u(L)}F$ (Table IV, 1st column), and l_i receives F_i at a rate of $\frac{u(l_i)}{u(L)}d^L_{min}$ from the seed (Table IV, 2nd column). At the same time, l_i sends F_i to l_j at a rate of $\frac{u(l_i)}{u(L)}d^L_{min}$ (Table IV, 4th column) and receives F_j at a rate of $\frac{u(l_j)}{u(L)}d^L_{min}$ from leecher l_j. Note that the total download rate of l_i is d^L_{min}.

In Case I, l_i sends F_i to l_j at the same rate that it obtains F_i from the seed. Therefore, when the process terminates after time F/d^L_{min}, all the leechers obtain the whole file chunk F_i. However, in some cases, l_j relies also on the seed to get F_i. Refer to Case II in the 2nd and 4th columns of Table IV. Before time T_0, $r_{si} > r_{ij}$, which means that l_i receives F_i faster than it distributes to leecher l_j. After T_0, l_i has obtained the whole F_i but l_j has not. At this time, the seed also helps l_j to obtain F_i by sending F_i at a rate specified in the 3rd column of Table IV. Due to space limitation, we refer interested readers to [4] for the detailed explanations of the whole scheduling mechanism.

B. Impact of Leecher Failure on the KR Algorithm

If leecher l_i leaves the system before the whole distribution process ends, the portion of F_i that l_i is responsible for distributing but has not yet sent to other leechers is affected. As only the seeds possess this portion after l_i has left, it has to distribute this remaining portion of F_i. There are two ways to handle this situation. One way is treating this affected portion as a new file, while another file distribution is launched

TABLE IV
RATE PROFILE OF LEECHER l_i

Case	Size of F_i [1st column]	Rate at which l_i downloads F_i from S (r_{si}) [2nd column]	Rate at which l_i downloads F_j from S [3rd column]	Rate at which l_i sends F_i to l_j (r_{ij}) [4th column]
I	$\frac{u(l_i)}{u(L)}F$	$\frac{u(l_i)}{u(L)}d^L_{min}$	0	$\frac{u(l_i)}{u(L)}d^L_{min}$
II	$\frac{F(d^L_{min}+\frac{u(l_i)-u(L)}{\lvert L\rvert-1})}{\lvert L\rvert d^L_{min}-u(L)}$	Before T_0, $d^L_{min}+\frac{u(l_i)-u(L)}{\lvert L\rvert-1}$; After T_0, 0. ($T_0=\frac{F}{\lvert L\rvert d^L_{min}-u(L)}$)	Before T_0, 0; After T_0, $\frac{d^L_{min}-u(l_j)}{\lvert L\rvert-1}$ ($T_0=\frac{F}{\lvert L\rvert d^L_{min}-u(L)}$)	$\frac{u(l_i)}{\lvert L\rvert-1}$
III	$(\frac{u(l_i)}{\lvert L\rvert-1}+\frac{u(S)}{\lvert L\rvert}-\frac{u(L)}{\lvert L\rvert(\lvert L\rvert-1)})\frac{F}{u(S)}$	Before T_0, $\frac{u(l_i)}{\lvert L\rvert-1}+\frac{1}{\lvert L\rvert}(u(S)-\frac{u(L)}{\lvert L\rvert-1})$; After T_0, 0. ($T_0=\frac{F}{u(S)}$)	Before T_0, 0; After T_0, $\frac{u(S)+u(L)}{\lvert L\rvert(\lvert L\rvert-1)}-\frac{u(l_j)}{\lvert L\rvert-1}$ ($T_0=\frac{F}{u(S)}$)	$\frac{u(l_i)}{\lvert L\rvert-1}$
IV	$\frac{u(l_i)}{u(L)}F$	$\frac{u(l_i)}{u(L)}u(S)$	0	$\frac{u(l_i)}{u(L)}u(S)$

using the remaining upload and download capacities of the seeds and leechers without affecting the distribution of other file chunks. Another approach is to collect the undistributed portions of all the file chunks as a new file and restart the distribution. The first approach is simpler, but it cannot reduce the distribution time. To illustrate, let us assume the leecher l where $d(l) = d^L_{min}$ leaves the system and $L' = L\backslash\{l\}$. We further assume that this configuration belongs to Case I. In Case I, $T_{min}(S, L)$ is F/d^L_{min}. After l has left, as all leechers are still using the rate of d^L_{min} to download the file, the download time cannot be reduced. On the other hand, if the undownloaded portions of the file chunks can be collected as a new file, as l has left, then $F/d^{L'}_{min}$ is smaller than $T_{min}(S, L)$, and the remaining peers can obtain the file faster.

A complete analysis involves four different cases of the two approaches, making up a total of eight variants. Due to space limitation, we only focus on the second approach because it makes the reduction of the download time possible. We start with Cases I and II. In these two cases, $T_{min}(S, L) = F/d^L_{min}$ and $u(S) \geq d^L_{min}$. Let $T'_{min}(S, L, F)$ be the total time needed to distribute the file in the case where a leecher fails.

1) Case I: Assume at time t, leecher l_i fails, and the current distribution process is suspended. A new file of size F', which consists of the undownloaded portion of the original file, is created and distributed to the remaining $|L| - 1$ leechers. Let $L' = L\backslash\{l_i\}$, thus $u(L') = u(L) - u(l_i)$. The minimum distribution time after reorganizing the distribution is

$$T'_{min}(S, L, F) = t + \frac{F'}{min\{u(S), d^{L'}_{min}, \frac{u(S)+u(L')}{|L|-1}\}}.$$

As

$$F' = F - \sum_{i=1}^{|L|} \frac{u(l_i)}{u(L)}(d^L_{min} * t) = F - d^L_{min} * t,$$

by the fact that $u(S) \geq d^L_{min}$,

$$\frac{F'}{u(S)} \leq \frac{F}{d^L_{min}} - t = T_{min}(S, L, F) - t.$$

As $d^{L'}_{min} \geq d^L_{min}$,

$$\frac{F'}{d^{L'}_{min}} \leq \frac{F}{d^L_{min}} - t = T_{min}(S, L, F) - t.$$

Therefore, the distribution process will not be prolonged if $\frac{u(S)+u(L')}{|L|-1}$ is larger than or equal to either $u(S)$ or $d^{L'}_{min}$. We now study if $\frac{u(S)+u(L')}{|L|-1}$ is the smallest among $u(S)$ and $d^{L'}_{min}$ and under what condition $T'_{min}(S, L, F)$ is larger than $T_{min}(S, L, F)$. For simplicity, we let d^L_{min} be d.

$$\begin{array}{lrcl} & \frac{F'}{\frac{u(S)+u(L')}{|L|-1}} + t & > & T_{min}(S, L, F) \\ \text{iff.} & \frac{F'(|L|-1)}{u(S)+u(L')} & > & T_{min}(S, L, F) - t \\ \text{iff.} & \frac{(F-d*t)(|L|-1)}{u(S)+u(L')} & > & \frac{F}{d} - t \\ \text{iff.} & \frac{(F-d*t)(|L|-1)}{u(S)+u(L')} & > & \frac{F}{d} - t \\ \text{iff.} & d * (|L| - 1) & > & u(S) + u(L') \\ \text{iff.} & u(l_i) & > & u(S) + u(L) - d * (|L| - 1) \end{array}$$

The above analysis shows that if $u(l_i) > u(S) + u(L) - d^L_{min} * (|L| - 1)$ and $min\{u(S), d^{L'}_{min}, \frac{u(S)+u(L')}{|L|-1}\} = \frac{u(S)+u(L')}{|L|-1}$, the distribution process will be prolonged after l_i fails.

2) Case II: We again let l_i leave at time t before the whole distribution process is complete, and $L' = L\backslash\{l_i\}$. Refering to Table IV, before time T_0, as $r_{sk} > r_{kj}$, leecher l_k possesses part of F_k that it has not sent to another leecher yet. Therefore, if the file distribution is suspended at this time due to the departure of leecher l_i, for each file chunk F_k where $k \neq i$, l_k possesses more of F_k than the other remaining leechers. This portion of F_k, which l_k has already obtained while other leechers have not, should not be collected

as a new file for distribution as a new file should contain the information that *all* leechers do not have. We refer the file portion of F_k that *some but not all* leechers possess as $F^k_{asyn}(t)$, where $k = 1, 2, ...|L| - 1$ and $k \neq i$. Each leecher l_k has its own unique $F^k_{asyn}(t)$. In summary, the original file consists of three parts: (1) the portion where every leecher has obtained (F_{done}), (2) the file portion that *some but not all* remaining leechers have ($F^k_{asyn}(t)$), and (3) the file portion that *no* remaining leecher possesses (F'). We now analyze the sizes of F_{done} and $F^k_{asyn}(t)$. There are two situations: $t < T_0$ and $t \geq T_0$. To simplify the notations, we overload F_{done} to refer to both the file chunk itself and the size of the file chunk. The same applies to $F^k_{asyn}(t)$ and F'.

When $t < T_0$

In this case, only l_i sends F_i to l_k, and the size of F_{done} is

$$F_{done} = \sum_{i=1}^{|L|} r_{ik} * t = \sum_{i=1}^{|L|} \frac{u(l_i)}{|L|-1} t = \frac{u(L)}{|L|-1} t \tag{9}$$

$$F^k_{asyn}(t) = (r_{sk} - r_{kj}) * t = (d^L_{min} - \frac{u(L)}{|L|-1}) * t \tag{10}$$

When $t \geq T_0$

In this situation, l_k has downloaded the whole file chunk F_k and thus $r_{sk} = 0$. l_k is still sending F_k to l_j at a rate of r_{kj}. Moreover, the seed is also sending F_k to l_j at a rate of r specified in the 3rd column of Table IV. Therefore,

$$\begin{aligned} F_{done} &= \sum_{i=1}^{|L|} r_{ik} * t + \sum_{i=1}^{|L|} \frac{d^L_{min} - u(l_i)}{|L|-1}(t - T_0) \\ &= \frac{|L| d^L_{min} t - F}{|L|-1} \end{aligned} \tag{11}$$

$$\begin{aligned} F^k_{asyn}(t) &= F_k - r_{kj} * t - r * (t - T_0) \\ &= \frac{F - d^L_{min} t}{|L|-1} \end{aligned} \tag{12}$$

From Eq. (10) and Eq. (12), $F^k_{sync}(t)$ is the same for every possible k. We use $F_{asyn}(t)$ to represent the collection of all the $F^k_{asyn}(t)$, and its size is $(|L| - 1) * F^k_{asyn}(t)$. Therefore,

$$F' = F - F_{done} - (|L| - 1) * F^k_{asyn}(t). \tag{13}$$

When we reorganize the distribution, we only have to consider $F_{asyn}(t)$ and F'. For F', we can apply the KR scheduling mechanism just as in Case I. On the other hand, we found that the situation where part of the file is possessed by some but not all leechers is similar to the circumstance in Case II after T_0. Hence, we can adopt the same technique to handle $F_{asyn}(t)$. We now analyze the following approach of distribution. After leecher l_i leaves, $F_{asyn}(t)$ is first distributed among the leechers so that every leecher finally obtains the whole $F_{asyn}(t)$. Then, F' is distributed as a new file.

In distributing $F_{asyn}(t)$, the objective is to synchronize all the file portions being held by all the remaining leechers. That is, we want to distribute $F^k_{asyn}(t)$ to leecher l_j where $j \neq i, k$. In Case II after time T_0, leecher l_k has already obtained the whole file chunk F_k and thus $r_{sk} = 0$. Therefore, the seeds and leecher l_k together distribute F_k to the other leechers. According to Case II in Table IV, the sum of the 3rd column and the 4th column represents the total download rate of a leecher l_j for receiving file chunk F_k after T_0, which is $\frac{d^L_{min}}{|L|-1}$. d^L_{min} is the bottleneck rate of the configuration, and l_j receives $|L| - 1$ file chunks from other leechers. Applying the same idea, when distributing $F^k_{asyn}(t)$, l_j receives at a rate of $\frac{1}{|L|-2} \min\{u(S), d^{L'}_{min}, \frac{u(S)+u(L')}{|L|-1}\}$. As all the $F^k_{asyn}(t)$ are of the same size, the distributions of the different $F^k_{asyn}(t)$ will finish at the same time. Let t_{sync} be the time needed in this phase and

$$t_{sync} = \frac{F^k_{asyn}(t)}{\frac{1}{|L|-2} \min\{u(S), d^{L'}_{min}, \frac{u(S)+u(L')}{|L|-1}\}}$$

The minimum distribution time after reorganizing the distribution is thus

$$T'_{min}(S, L, F) = t + t_{sync} + \frac{F'}{\min\{u(S), d^{L'}_{min}, \frac{u(S)+u(L')}{|L|-1}\}}.$$

That is,

$$T'_{min}(S, L, F) = t + \frac{(|L| - 2)F^k_{asyn}(t) + F'}{\min\{u(S), d^{L'}_{min}, \frac{u(S)+u(L')}{|L|-1}\}}. \tag{14}$$

By Eq. (13),

$$(|L| - 2)F^k_{asyn}(t) + F' = F - F_{done} - F^k_{asyn}(t). \tag{15}$$

When $t < T_0$, by Eqs. (9) and (10),

$$(|L| - 2)F^k_{asyn}(t) + F' = F - d^L_{min} t.$$

When $t \geq T_0$, by Eqs. (11) and (12),

$$(|L| - 2)F^k_{asyn}(t) + F' = F - d^L_{min} t \qquad \text{as well.}$$

Following the same explanation in Case I, if $u(l_i) > u(S) + u(L) - d^L_{min} * (|L| - 1)$ and $min\{u(S), d^{L'}_{min}, \frac{u(S)+u(L')}{|L|-1}\} = \frac{u(S)+u(L')}{|L|-1}$, the distribution process will be prolonged after l_i fails.

3) Case III: This case is similar to Case II in the sense that $r_{si} > r_{ij}$. Therefore, when the distribution is suspended, as in Case II, we have F_{done}, $F^k_{asyn}(t)$, and F'.

When $t < T_0$

$$F_{done} = \sum_{i=1}^{|L|} r_{ik} * t = \sum_{i=1}^{|L|} \frac{u(l_i)}{|L|-1} t = \frac{u(L)}{|L|-1} t \tag{16}$$

$$F^k_{asyn}(t) = (r_{sk} - r_{kj}) * t = \frac{1}{|L|}(u(S) - \frac{u(L)}{|L|-1}) * t \tag{17}$$

When $t \geq T_0$

$$\begin{aligned} F_{done} &= \sum_{i=1}^{|L|} r_{ik} * t + \sum_{i=1}^{|L|} (\frac{u(S) + u(L)}{|L|(|L|-1)} - \frac{u(l_i)}{|L|-1})(t - T_0) \\ &= \frac{(u(S) + u(L))t - F}{|L|-1} \end{aligned} \tag{18}$$

$$F^k_{asyn}(t) = F_k - r_{kj} * t - r * (t - T_0)$$
$$= \frac{1}{|L|-1}(F - \frac{u(S)+u(L)}{|L|}t) \quad (19)$$

Eqs. (13), (14), and (15) still apply. Therefore, for any t,

$$(|L|-2)F^k_{asyn}(t) + F' = F - \frac{u(S)+u(L)}{|L|}t$$

In this case, if $u(l_i) > \frac{u(S)+u(L)}{|L|}$ and $min\{u(S), d^{L'}_{min}, \frac{u(S)+u(L')}{|L|-1}\} = \frac{u(S)+u(L')}{|L|-1}$, the distribution process will be prolonged after l_i fails.

4) Case IV: In this case, it is not possible for the performance to improve. We prove by contradiction and assume to the contrary that $T_{min}(S,L,F) > T'_{min}(S,L,F)$. Then,

$$\frac{F'}{u(S)} + t > \frac{F'}{\min\{u(S), d^{L'}_{min}, \frac{u(S)+u(L')}{|L|-1}\}} + t$$
$$\Rightarrow \quad \frac{F'}{u(S)} > \frac{F'}{\min\{u(S), d^{L'}_{min}, \frac{u(S)+u(L')}{|L|-1}\}}$$
$$\Rightarrow \quad u(S) < \min\{u(S), d^{L'}_{min}, \frac{u(S)+u(L')}{|L|-1}\}. \quad (20)$$

Eq. (20) is not possible because the left hand side of the inequality cannot be smaller than $u(S)$.

C. Leecher Failure in Grouping

We now use an example to show that when grouping is applied, and the leecher with the highest upload bandwidth leaves before the distribution ends, the *average download time* can still be smaller than that in the case where no grouping is applied. Tables V and VI show the bandwidth configuration of a system with three seeds and six leechers.

TABLE V
UPLOAD BANDWIDTH OF SEEDS

Peers	s_1	s_2	s_3
$u(s_i)/Kbps$	150	200	250

TABLE VI
UPLOAD AND DOWNLOAD BANDWIDTH OF LEECHERS

Peers	l_1	l_2	l_3	l_4	l_5	l_6
$u(l_i)/Kbps$	50	50	50	150	150	250
$d(l_i)/Kbps$	150	150	250	300	400	350

From Tables V and VI, we have $u(S) = 600$ Kbps, $u(L) = 700$ Kbps, $\frac{u(L)+u(S)}{|L|} = 216.7$ Kbps, and $d^L_{min} = 150$ Kbps. As $d^L_{min} < \min\{u(S), \frac{u(L)+u(S)}{|L|}\}$ and $d^L_{min} > \frac{u(L)}{|L|-1}$, the configuration falls in Case II as discussed before. As $u = u(S)+u(L)-(|L|-1)d^L_{min} = 550$ Kbps and $u(l_6) < u$, we know that the system performance will not be degraded if l_6 fails during the file distribution process. After applying the calculation described in the previous section, we find that $T_{min}(S,L,F) = T^{KR}_{avg}(S,L,F) = 2000$ s, so that the performance remains unchanged even if l_6 fails when grouping is not adopted.

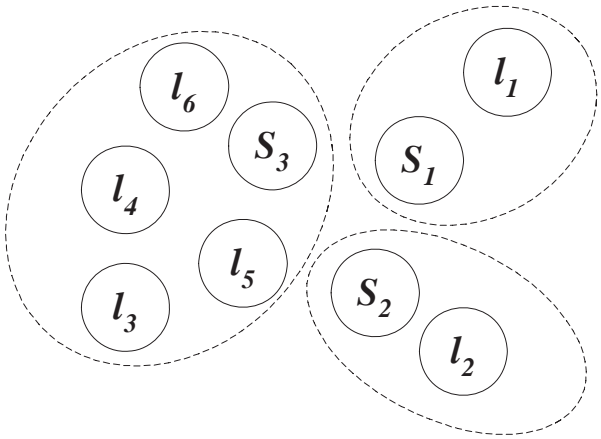

Fig. 3. Grouping result using the seeds and leechers in Tables V and VI.

The configuration is feasible in grouping. If we apply the grouping protocol, we have the following grouping result illustrated in Fig. 3: $G_1 = \{s_1, l_1\}, G_2 = \{s_2, l_2\}$ and $G_3 = \{s_3, l_3, l_4, l_5, l_6\}$.

We now assume l_6 of G_3 fails. In G_3, $u(S_3) = 250$ Kbps, $u(L_3) = 600$ Kbps, $\frac{u(S_3)+u(L_3)}{|L_3|} = 212.5$ Kbps, $d^{L_3}_{min} = 250$ Kbps. As $\frac{u(S_3)+u(L_3)}{|L_3|} < \min\{u(S_3), d^{L_3}_{min}\}$, the bandwidth characteristics of G_3 fall in Case III in the above section. As $u = \frac{u(S_3)+u(L_3)}{|L_3|} = 212.5$ Kbps, the failure of l_6 will cause the system performance to suffer because $u(l_6) > u$. After calculation, $T_{min}(S_3, L_3, F) = 1493.75$ s and $T^G_{avg}(S,L,F) = 1662.5$ s. From the results above, we can see that although the performance in the grouped case suffers after l_6 fails, $T^G_{avg}(S,L,F)$ is still less than $T^{KR}_{avg}(S,L,F)$ when grouping is not applied. Hence, grouping can bring about benefits even when the leecher with the largest bandwidth fails.

Moreover, when grouping is used, reorganization occurs only within the group that is affected, which is much simpler compared with the redistribution of the whole system. In fact, by grouping, reorganization becomes scalable because even when there are many seeds and leechers, our grouping mechanism will partition them into groups of small sizes. The effect of leecher failure is then isolated to a smaller group only.

VI. PERFORMANCE EVALUATION

We conduct extensive simulations to evaluate the performance improvement of our grouping mechanisms. We assume the grouping is calculated by a coordinator seed as mentioned in Section IV-C, and no peer leaves before the distribution ends. Therefore, each seed/leecher only needs to send one message to the coordinator seed to report its bandwidth capacity, making up a total of $O(|S| + |L|)$ messages. To measure the distribution time performance of our grouping mechanism, we use ρ as defined in Eq. (2). The ratio means *total time to distribute the file/average time to distribute the file*. We use the same ratio for the BT-like algorithm. However, because the BT-like algorithm is not a perfect algorithm, the

total time for distributing the file must not be smaller than the time it takes for our grouping mechanism.

A. Simulation Setup

We vary the size of S, the size of L, and the upload and download capacities of the peers to observe the performance. Table VII summarizes the simulation parameters.

TABLE VII
SIMULATION PARAMETERS

$\|S\|$	$[1, 20]$
$\|L\|$	$[20, 140]$
$u(s)$	$[56Kbps, 12Mbps]$
$d(l)$	$[36Kbps, 12Mbps]$

We set the lower bound of the seed upload limit to 56 Kbps because this estimates the speed of the dial-up service. On the other hand, 12 Mbps is close to the speed available in ADSL2 [29]. The lower bound of $d(l)$ is smaller than the lower bound of $u(s)$ because not all peers connect to high-speed dial-ups. We use different ranges of $u(l)$ to evaluate our protocol. Basically, $u(l)$ is in the range of $[p, p*q]$, where $p = 20$ Kbps, which is also the lower limit used in [23]. We select q to be 50, 100, and 180 to study the performance when the upload capacities of the leechers are of different degrees of diversities. Note that the upload capacities of leechers are smaller than their download bandwidth to simulate the current widely used ADSL technology. Each data point in every figure is the average result of 30 different configurations. Note that because we generate configurations randomly, it is possible that some configurations are not feasible in grouping, as defined in Section III-B. That is, grouping does not bring about benefits in these configurations. To evaluate the effect of grouping more directly, we ignore the overheads in setting up the TCP connections among peers and in the packet headers.

For comparison, we develop a BT-like algorithm. There are 100 file pieces. Initially, leechers do not have any file piece, and each leecher makes a request for a randomly selected file piece. Afterwards, the rarest-piece-first mechanism is used to select the file piece to request. The request is sent to a randomly selected peer. Each peer serves one request at a time, and it can download one piece at most at any time. Each peer buffers one pending request at most. That is, when a request arrives when a peer is serving another peer, if the buffer is empty, the peer receiving this request puts the request in the buffer. If there is another request in the buffer, this newly arrived request is rejected. If the request it sends is rejected, the leecher waits for time t before sending the same request to another peer. t is selected uniformly from $[0s, 1s]$.

B. Simulation Results

Figs. 4 and 5 show the performance improvement ratios ρ of our mechanism and the BT-like mechanism from different viewing perspectives of the same 3-D plot. Fig. 4 demonstrates the performance when there are more than 100 leechers while Fig. 5 provides the results when there are around 20 leechers. The larger ρ, the better the performance. The first three planes

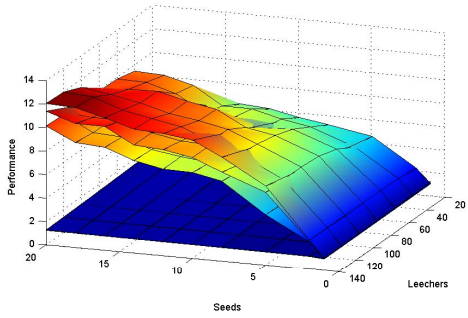

Fig. 4. Performance comparison of BitTorrent and Greedy Grouping over varying q, number of seeds and leechers, view 1. From top to bottom: Greedy grouping with q = 180, 100 and 50, and BitTorrent with q = 180, 100, and 50.

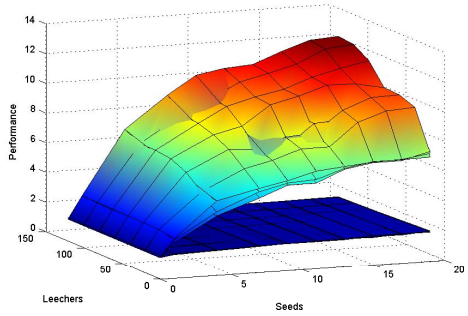

Fig. 5. Performance comparison of BitTorrent and Greedy Grouping over varying q, number of seeds and leechers, view 2. From top to bottom: Greedy grouping with q =180, 100, and 50, and BitTorrent with q =180, 100, and 50.

from the top are the results of our algorithm when q are 180, 100, and 50, respectively. The lower planes are the results of the BT-like mechanism. Since they are too close to each other in Figs. 4 and 5, we put them separately in Figs. 6 and 7 for better illustration.

Our simulation results show that the Greedy leecher grouping algorithm outperforms the BT-like mechanism. The improvement is very significant when the system consists of many seeds and leechers. We now describe the relations between ρ and different parameters.

Number of seeds

From Fig. 4, it can be observed that the performance of our protocol is better when there are more seeds. There are several reasons. First, when there are more seeds, it is more likely that $u(S) > d^L_{min}$, which means that it is more likely that the configuration is feasible in grouping. On the other hand, given more seeds, the configuration can be partitioned into more groups and the effect of slow leecher can be isolated to a smaller group. Fewer leechers will be affected by this slow leecher and the *average download time* can be reduced.

Number of leechers

ρ also increases when there are more leechers. Note that grouping cannot be applied unless d^L_{min} is small enough to become a bottleneck, as shown in Section III-B. When grouping is not applied, our mechanism does not bring about

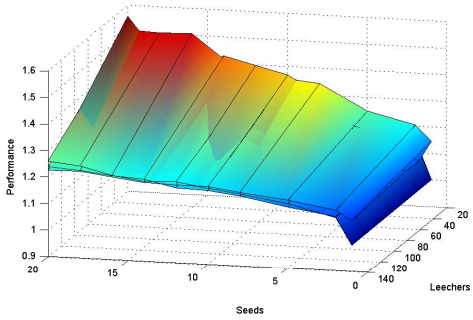

Fig. 6. Performance of BitTorrent over varying q, number of seeds and leechers, view 1. From top to bottom: BitTorrent with q = 180, 100, and 50.

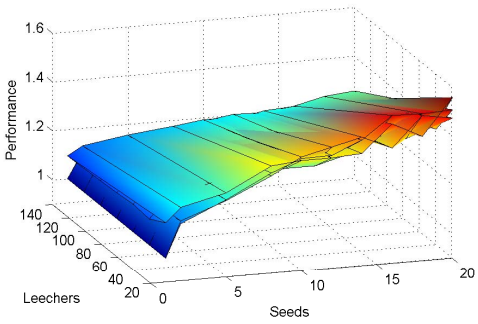

Fig. 7. Performance of BitTorrent over varying q, number of seeds and leechers, view 2. From top to bottom: BitTorrent with q =180, 100, and 50.

benefits and cannot reduce the average download time. When there are more leechers, it is more likely that the configuration is a feasible one as there is a higher chance that there is a leecher with a small download capacity. Therefore, the performance increases when the number of leechers increases.

Diversity of leecher upload capacities

We generate leecher upload capacities from the range of $[p, p * q]$, where p is set to 20 Kbps. Larger q implies a higher diversity among the leecher upload bandwidth, and the performance becomes better in general. However, we observe from Figs. 4 and 5 that the difference in performance is larger when there are more leechers. To understand this behavior, we study the relation between number of leechers and the value of $\frac{u(S)+u(L)}{|L|}$, as shown in Fig. 8. It can be observed that $\frac{u(S)+u(L)}{|L|}$ is larger when q is larger. If $T_{min}(S,L) = \frac{|L|F}{u(S)+u(L)}$, a larger q implies larger $\frac{u(S)+u(L)}{|L|}$ and a smaller $T_{min}(S,L)$.

When $|L|$ is large, it is very likely that the configuration is feasible, and grouping can be applied. After partitioning S and L into smaller groups, these small groups (S_i, L_i) very often have a $T_{min}(S_i,L_i) = \frac{|L_i|F}{u(S_i)+u(L_i)}$. Therefore, a larger q would reduce the average download time. However, when $|L|$ is small, and many configuration are not feasible in grouping, the effect of q would not be obvious.

Performance of the BT-like mechanism

Figs. 4 and 5 demonstrate that our protocol outperforms the

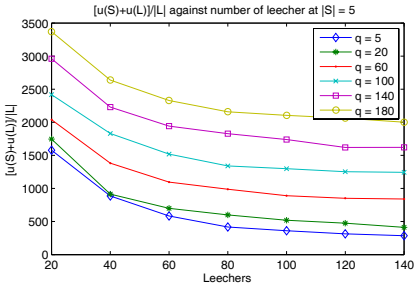

Fig. 8. Relation between $|L|$ and $\frac{u(S)+u(L)}{|L|}$.

BT-like mechanism in all configurations. Moreover, Figs. 6 and 7 show a similar trend as in our protocol when the number of seeds increases. However, the BT-like mechanism has better performance when there are fewer leechers, and this is different from our grouping algorithm. When the number of leechers increases, it is much more likely that the minimum download time is $\frac{F|L|}{u(S)+u(L)}$. From Fig. 8, $\frac{u(S)+u(L)}{|L|}$ is smaller when there are more leechers in the system. Therefore, if $T_{min}(S,L) = \frac{u(S)+u(L)}{|L|}$, $T_{min}(S,L)$ will increase when the number of leechers increases and the performance suffers.

In summary, our simulation results show that our protocol can effectively reduce the average download time of a wide range of configurations. Our mechanism is particularly useful when there are many seeds and leechers in the system.

VII. Conclusion and Future Works

P2P content distribution networks have been largely employed for file distributions over the Internet. In evaluating the performance of such systems, file distribution time is an important measure. In this paper, we first briefly introduce the theoretical minimum time needed for file distribution under the fluid model derived by Kumar and Ross. Next, we study the factors affecting the distribution time. Subsequently, we propose the greedy grouping mechanism to reduce further the *average download time* without prolonging the *minimum download time*. Our scheme organizes the seeds and the leechers based on their upload and download bandwidth characteristics. The performance is enhanced by isolating slow leechers and reducing their negative effects on other leechers. Simulation results are supplied to illustrate the performance under a wide range bandwidth capacities.

The advantages brought by grouping are not limited to what we have mentioned above. We then demonstrate the robustness of grouping in the event of leecher failure. The impact of leecher failure is contained to a fraction of the leechers. We propose two strategies to handle leecher failure situations. One way is to treat the affected file portion as a new file and launch a new file distribution using the remaining resources. This way, the distribution of other file chucks are unaffected. Another way is to combine the undistributed partitions as a new file and start a new file distribution. Subsequently, we derive the conditions in which the performance improves after a leecher fails for different cases.

In summary, this paper studies how to reduce the average download time and the grouping strategy from a theoretical

perspective. In the future, we would like to investigate how to apply the theoretical results to a practical situation where nodes are more dynamic, and peers may possess some part of the file when they join the network. On the other hand, grouping can work complimentarily with many other file sharing mechanisms. Our preliminary results show that our grouping mechanism also works well with the tit-for-tat strategy of the BT protocol to reduce further the average download time. We would like to conduct more extensive studies in this direction in our future works.

ACKNOWLEDGMENTS

The preliminary version of this paper appeared in the Proceedings of the IEEE International Conference on Communications (ICC), 2008 [30].

REFERENCES

[1] B. Cohen, "Incentives build robustness in bittorrent," in *Proc 1st Workshop Economics Peer-to-Peer Systems*, 2003.
[2] PPLive Official Web Site, [Online.] Available: http://www.pplive.com/en, 2007.
[3] D. Qiu and R. Srikant, "Modeling and performance anaysis of bittorrent-like peer-to-peer networks," in *Proc. ACM SIGCOMM*, 2004, pp. 367-378.
[4] R. Kumar and K. Ross, "Peer assisted file distribution: The minimum distribution time," in *Proc. IEEE Workshop Hot Topics Web Systems Technol.(HOTWEB2006)*, pp. 1-11.
[5] G. Mundinger, R. Webb, and G. Weiss, "Analysis of peer-to-peer file dissemination," in *Proc. 8th Workshop Mathematical Performance Modeling Analysis(MAMA2006)*, pp. 12-14.
[6] G. Mundinger, R. R. Webb, and G. Weiss, "Analysis of peer-to-peer file dissemination amongst users of different upload capacities," in *Performance 2005 Issue*, vol. 34, pp. 5-6.
[7] J. S. K. Chan, V. O. K. Li, and K.-S. Lui, "Performance comparison of scheduling algorithms for peer-to-peer collaborative file distribution," *IEEE J. Sel. Areas Commun.*, vol. 25, pp. 146-154, Jan. 2007.
[8] S. Sanghavi, B. Hajek, and L. Massoulie, "Gossiping with multiple messages," in *Proc. IEEE Infocom 2007*, pp. 2135-2143.
[9] C. Gkantsidis and P. R. Rodriguez, "Network coding for large scale content distribution," in *Proc. Infocom 2005*, pp. 2235-2245.
[10] R. Ahlswede, N. Cai, S. Y. R. Li, and R. W. Yeung, "Network information flow," *IEEE Trans. Inf. Theory*, vol. 46, pp. 1204-1216, July 2000.
[11] E. W. Biersack, P. Rodriguez, and P. Felber, "Performance analysis of peer-to-peer networks for file distribution," in *Proc. 5th International Workshop Quality Future Internet Services (QofIS'04)*, 2004, pp. 1-10.
[12] A. R. Bharambe, C. Herley, and V. N. Padmanabhan, "Analyzing and improving a bittorrent network's performance mechanisms," in *Proc. Infocom 2006*, pp. 1-12.
[13] F. Clevenot, P. Nain, and K. Ross, "Stochastic fluid models for cache clusters," Technical Report 4815, INRIA, Sophia Antipolis, 2003.
[14] F. Clevenot and P. Nain, "A simple fluid model for the analysis of the squirrel peer-to-peer caching system," in *Proc. IEEE Infocom 2004*, pp. 86-95.
[15] R. Gaeta, M. Gribaudo, D. Manini, and M. Sereno, "Fluid stochastic petri nets for computing transfer time distribution time in p2p file sharing applications," *Electronic Notes Theoretical Computer Science, Elsevier*, vol. 128, pp. 79-99, Apr. 2005.
[16] D. Manini and M. Gribaudo, "Modelling search, availability, and parallel download in p2p file sharing applications with fluid model," in *Proc. International Conf. Advanced Computing Commun.(ADCOM2006)*, pp. 449-454.
[17] K. Kojima, "Grouped peer-to-peer networks and self-organization algorithm," in *Proc. IEEE International Conf. Systems, Man Cybernetics*, 2003, pp. 2970-2976.
[18] Y. He, J. Zhang, X. Niu, and Q. Zhao, "Search and index in locality-based clustering overlay," in *Proc. IEEE Internatioanl Symp. Cluster Computing Grid*, 2005, pp. 229-236.
[19] X. Bai, S. Liu, P. Zhang, and R. Kantola, "ICN: interest-based clustering network," in *Proc. 4th International Conf. Peer-to-Peer Computing*, 2004, pp. 219-226.
[20] The Freenet Project Official Website, [Online.] Available: http://freenetproject.org/, 2007.
[21] P. Fraigniaud, P. Gauron, and M. Latapy, "Combining the use of clustering and scale-free nature of user exchanges into a simple and efficient p2p system," in *Proc. 11th International Conf. Euro-Par*, 2005.
[22] Y. Busnel and A.-M. Kermarrec, "Proxsem: interest-based proximity measure to improve search efficiency in p2p systems," in *Proc. ECUMN*, 2007.
[23] A. Legout, N. Loigkas, E. Kohler, and L. Zhang, "Clustering and sharing incentives in bittorrent systems," in *Proc. ACM SIGMETRICS*, 2007.
[24] Ono Plugin for the Vuze (Azureus) BitTorrent Client, [Online.] Available: http://www.aqualab.cs.northwestern.edu/projects/Ono.html, 2007.
[25] M. Castro, P. Druschel, A.-M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh, "Splitstream: high-bandwidth multicast in a cooperative environment," in *Proc. 19th ACM Symp. Operating Systems Principles (SOSP)*, 2003.
[26] J. D. Mol, D. H. Epema, and H. J. Sips, "The orchard algorithm: building multicast trees for p2p video multicasting without free-riding," *IEEE Trans. Multimedia*, vol. 9, no. 8, pp. 1293-1604, Dec. 2007.
[27] V. Venkataraman, K. Yoshida, and P. Francis, "Chunkyspread: Hheterogeneous unstructured end system multicast," in *Proc. 14th IEEE International Conf. Netw. Protocols*, 2006.
[28] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 2nd ed. MIT Press, 2002.
[29] International Telecommunication Union Official Website, [Online.] Available: http://www.itu.int/rec/t-rec-g.992.3/en, 2009.
[30] L. Ma, X. Wang, and K.-S. Lui, "A novel peer grouping scheme for p2p file distribution networks," in *Proc. IEEE International Conf. Commun.(ICC)*, 2008, pp. 5598-5602.

Lingjun Ma received the Bachelor of Engineering degree in Information Engineering from Tsinghua University, China in 2006. He then received the Master of Philosophy degree in Electrical and Electronic Engineering from the University of Hong Kong in 2008. His research interests include Peer-to-Peer networking and its application in file distribution.

Pui-Sze Tsang received the BEng degree in Information Engineering from the University of Hong Kong (HKU), Pokfulam, Hong Kong in 2008. She is now working towards the MPhil degree in Electrical and Electronic Engineering at HKU. Her research interests include peer-to-peer networking, computer systems and network protocols design.

King-Shan Lui (S'00-M'03-SM'09) received the B.Eng. and M.Phil. degrees in Computer Science from the Hong Kong University of Science and Technology. After receiving her Ph.D. degree from the University of Illinois at Urbana-Champaign, USA, she joined the Department of Electrical and Electronic Engineering of the University of Hong Kong. Her research interests include network protocol design and analysis, sensor networks, and Quality-of-Service issues.