

Performance Comparison of Scheduling Algorithms for Peer-to-Peer Collaborative File Distribution

Jonathan S K Chan, Victor O K Li, and King-Shan Lui

Abstract—Peer-to-Peer file sharing applications in the Internet, such as BitTorrent, Gnutella, etc., have been immensely popular. Prior research mainly focuses on peer and content discovery, overlay topology formation, fairness and incentive issues, etc. However, little attention has been paid to investigate the data distribution problem which is also a core component of any file sharing application. In this paper, we present the first effort in addressing this collaborative file distribution problem and formally define the scheduling problem in a simplified context. We develop several algorithms to solve the problem and study their performance. We deduce a theoretical bound on the minimum download time experienced by users and also perform simulations to evaluate our algorithms. Simulation results show that our graph-based *dynamically weighted maximum-flow* algorithm outperforms all other algorithms. Therefore, we believe our algorithm is a promising solution to be employed as the core scheduling module in P2P file sharing applications.

Index Terms—Peer-to-Peer; P2P; file sharing; data distribution; scheduling algorithms

I. INTRODUCTION

Peer-to-Peer (P2P) applications have become immensely popular in the Internet. One of the most popular applications of P2P networks is collaborative sharing of large video/audio files and software. Traditional methods for file sharing, such as the client/server approach (e.g. FTP, WWW), suffer from scalability bottleneck. As the outgoing bandwidth of the server is shared among all simultaneous clients, the more the clients, the less bandwidth each client can have. Hence, the performance of client/server approach deteriorates rapidly as the number of simultaneous clients increases. P2P file sharing solves the problem by allowing peers to act as servers. Interestingly, in a well-designed P2P file sharing network, more peers participating in the file sharing session generally means better performance, as each peer could download simultaneously from multiple peers. Due to the significant performance improvements with collaborative file sharing, there has been widespread use of P2P file sharing applications like BitTorrent [1], Gnutella [2], Kazaa [3], Napster [4], etc., just to name a few. Interestingly a recent study [5] shows that BitTorrent traffic accounts for an amazing 35% of all Internet traffic, which is more than any other single protocol such as the HTTP, POP and

SMTP, demonstrating the increasing importance of P2P file sharing systems among the Internet community.

Experiments, such as those in [6], have shown that parallel downloading scheme in P2P file sharing systems could result in higher aggregate download rate and thus shorter download time. In parallel downloading, an end user opens multiple connections with multiple file sources to download different portions of the file from different sources and then reassembles the file locally. According to the analysis in [7], the service capacity increases greatly compared with schemes that share the file as a whole.

P2P file sharing systems have attracted much attention for the past few years since its recent inception. Although there are many papers related to P2P systems investigating issues of diversified interest, most of them focus on topics like overlay topology formation, peer discovery, content search, sharing fairness, incentive mechanisms, etc. One of the core components of file sharing systems, the data distribution scheduling problem, on the other hand, has received little attention. The scheduling problem is important because it governs how file pieces are transmitted and distributed among peers, and has a direct effect on the performance. A poor data distribution schedule could result in considerably longer download time, while a good schedule could shorten the completion time and efficiently utilize all resources like network bandwidth. This article studies the “data distribution scheduling problem” in P2P collaborative file sharing systems and proposes a novel maximum flow graph model for efficiently solving the scheduling problem.

The major contributions of this paper are summarized as follows:

- We adopt the push-based decision model instead of the commonly used pull-based model. Our push-based model reduces message overheads by eliminating the need of transmitting file piece request messages.
- We formally define the data distribution scheduling problem using matrix formulations with granularity down to each file piece possessed by each peer for the case when peers have asymmetric upload and download bandwidths. We also derive a theoretical lower bound of the transmission time required.
- We develop several algorithms (including a novel graph-based dynamically weighted maximum-flow algorithm) for determining the file piece distribution schedule, and evaluate their performance by simulations.

This research is supported in part by the Areas of Excellence Scheme, established by the University Grants Committee, Hong Kong Special Administrative Region, China, Project No. AoE 99-01.

- Simulation results show that our dynamically weighted maximum-flow algorithm outperforms all other algorithms.

The rest of this paper is organized as follows. Section II briefly describes some related work. The communication model, notations and analysis are given in Section III. Section IV presents various algorithms for approaching the scheduling problem, followed by an evaluation using computer simulations in Section V. Some of our future research directions are presented in Section VI and we conclude the paper in Section VII.

II. RELATED WORK

Due to the dramatic increase of broadband user population, there has been large-scale deployment of P2P file sharing systems in the Internet. One of the most representative P2P file sharing applications is BitTorrent (BT) [1], [8], which has thousands of simultaneous users. Tracker servers maintain a list of participating peers. When a new peer joins, the tracker server will send the new peer the contact information of a few connected peers from the list. Then, this new peer contacts these peers directly for downloading the file.

A shared file is partitioned into multiple small pieces, usually about 256Kbytes or 512Kbytes in size. Peers exchange information on which pieces they currently possess, and request missing pieces from others. A peer can maximize its download speed by requesting different pieces from different peers at the same time. A scheduling mechanism is needed for a peer to decide which pieces to request and to whom such a request should be made. A poor scheduling algorithm may lead to every peer getting nearly the same set of pieces and consequently decreases the number of file piece sources which a peer can simultaneously download from. BT employs the *Rarest Element First (REF)* algorithm, in which those pieces that most peers do not have are downloaded first. This algorithm is good at increasing the availability of different file pieces in the network and is efficient in distributing all pieces from the original source to different peers across the network. However, our simulation results show that *REF* is not an optimal scheduling algorithm.

The self-scaling properties of collaborative file distribution architectures, where each peer has equal upload and download rates of b , and there are no transmission failures have been shown in [9]. If the file size is f , the file is chopped into m pieces, and let $\tau = f/b$ be the time needed for transmitting the whole file,

it would take $Time = 1 + \lceil \log_k N \rceil \frac{k\tau}{m}$ to serve all the N peers organized in k spanning trees. The time required increases only logarithmically with increasing number of peers N in collaborative distribution, as opposed to a linear increase in the case of client-server distribution strategies.

Two crucial factors that affect the global effectiveness of the file distribution process are evaluated in [10]. They have direct impacts on the delay experienced by the peers and the global throughput of the system and they are:

- Peer selection strategies: random, least missing, most missing, adaptive missing
- File piece selection strategies: random, rarest

It is found that the most missing peer selection strategy, in which the peer with the largest number of un-received file pieces is selected for transmission first, can minimize the download time of the last complete peer and produce a smaller variance of the download times of the peers. This is because the most missing peer selection strategy ensures regular progress of all the peers by having an even dissemination of the file pieces. On the other hand, for file piece selection strategies, it is concluded that the rarest piece first strategy shows significant performance improvement over the random piece selection strategy. Some of our algorithms presented in Section IV are basically in the same spirit with the evaluations presented in [10] by employing the most missing peer selection and the rarest file piece selection strategies. We further develop a novel maximum-flow algorithm that ensures maximum number of transmissions can be scheduled while taking into account the rarity of file pieces and the demand for file pieces of the peers.

The problem of broadcasting or multicasting a single message in heterogeneous networks has been investigated in a number of papers [11], [12], [13], [14] from the algorithmic point of view. In particular, [11] studies the problem in a network where nodes have different processing times and the transmission times between different node pairs also vary. The authors evaluate the completion time of various algorithms and show that the well-known *Fastest Node First* algorithm may result in solutions which are worse than the optimal by an unbounded factor. They subsequently propose the *Fastest Edge First* and *Earliest Completing Edge First* algorithms to better solve the problem. Reference [12] studies the problem in a similar network as [11] except that all transmission times are assumed to be the same. It proves that the problem of minimizing the maximum completion time of broadcasting a single message is NP-hard. It also shows the *Fastest Node First* heuristic in computing broadcast schedules can produce an 1.5 approximation schedule for the same problem. It is worth noting that the above papers [11]–[14] only analyze the problem of distributing one single message. In P2P file sharing systems, a file is divided into multiple file pieces instead of one big piece. Therefore, efficient scheduling algorithms have to be developed and this is the aim of our work. To the best of our knowledge, this paper presents the first effort in addressing this problem based on some simplifications.

Some measurement-based research [15], [16] has been conducted to study the performance of P2P systems. In [15], the authors reveal the inherent difficulty of the peer selection problem and recognize the selection of peers would have profound effect on the overall performance. They tried to use some bandwidth probing techniques to optimize the selection of “good” peers. On the other hand, some analytical models [7], [17] have been proposed. In [7], a branching process and a Markov chain model are proposed to study the transient regime and the steady state regime of the BitTorrent system,

respectively. A fluid model is presented in [17], where the expressions of the numbers of incomplete peers and seeds (completed peers) could be obtained from the parameters such as the peer arrival rate, departure rate and the upload/download rates. However, previous models just consider the network, a particular peer or a particular file-sharing session as a whole, while our work is the first to model the data traffic transmission as a matrix representation down to the granularity of each file piece possessed by each peer. We also care about the scheduling selection of “good” peers, as well as “good” file pieces.

We have done a preliminary study in [18] with a homogeneous network assumption, where all the peers have symmetric upload and download bandwidths and the transmission time for sending a message from any node to any other node is the same. In this special case, we proposed a weighted *Bipartite Matching (BPM)* algorithm, which is a graph-based algorithm that can efficiently and optimally solve all the cases tested. In this paper, we are going to extend our results to networks with asymmetric bandwidth allocation and develop algorithms that can efficiently solve the problem.

III. PROBLEM DEFINITION

We base our study on the following communication model:

A. Communication Model

We assume users are situated at the edge of the network, with logical links connecting every pair of peers (i.e. fully-connected graph, an example network is shown in Fig. 1).

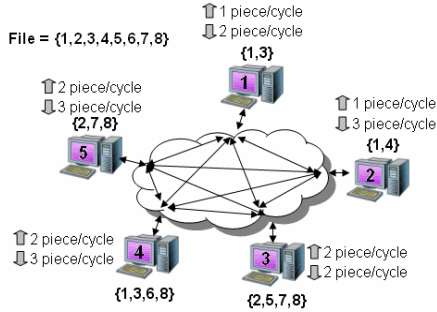


Fig. 1. Fully-connected network with asymmetric bandwidth allocation

1) *Synchronous Scheduling*: Due to the initial complexity we face when first studying this problem, we begin our investigation by assuming the transmission time for sending a message from any node to any other node is the same. That is, the data distribution scheduling is performed in discrete cycles synchronously.

2) *Exchange of Information*: As in BitTorrent, each peer actively communicates with other actively connected peers to announce his own set of possession information about which file pieces he currently possesses. This possession information is sent in the form of a bit-vector, say for example Peer 4 will announce $\{1,0,1,0,0,1,0,1\}$ to others, meaning that he currently possesses file pieces 1, 3, 6, and 8. For a typical video file of 650Mbytes with each file piece having a size of 512Kbytes, we will have 1300 bits (~ 163 bytes) in the bit-vector, which is small and will not consume too much bandwidth relative to the actual

transmission of the data file piece itself. In the following sections in this paper, these possession bit-vectors are assumed to be exchanged without errors between all connected peers before the scheduling of the next cycle.

3) *Asymmetric Bandwidth*: Each peer may have asymmetric upload and download bandwidths. This is common in the Internet where most users connect to the network using the Asymmetric Digital Subscriber Line (ADSL) technology. We assume the nodes can only send or receive an integral number of file pieces for each cycle. An example is shown in Fig. 1, where Peer 1 can send out one file piece and receive two pieces from others for each cycle, and Peer 2 can send one file piece and receive three pieces for each cycle, etc.

4) *Quasi-Static Assumption*: In this paper, we assume all the peers will not leave the file sharing session once initiated until the distribution process completes. Although this assumption is unlikely to hold in real systems, where peers may come and go and the peers may switch to exchange with different sets of peers during the distribution process, we could reasonably extend the notion of this quasi-static assumption to represent a particular period, say one minute, during which the peers keep active exchanges with this particular set of peers and we aim at transmitting the largest amount of data during this period by employing our proposed scheduling algorithms.

5) *Notations and Definitions*: Let the number of participating peers be N and the number of file pieces be M . The shared file F is chopped into M smaller pieces $F = \{F_1, F_2, \dots, F_M\}$ and each peer possesses a subset of F . We represent the file piece possession information as an $N \times M$ matrix P , called the *possession matrix*. $P_{ij} = 1$ if and only if node i possesses file piece F_j ($1 \leq i \leq N, 1 \leq j \leq M$); otherwise $P_{ij} = 0$. We use P^t to denote the possession matrix at time t . Refer to Fig. 1 where $F = \{F_1, F_2, \dots, F_8\}$ and $\{\dots\}$ next to a node indicates the pieces that the node possesses, the possession matrix at the beginning ($t = 0$) is:

$$P^0 = \begin{pmatrix} 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 \end{pmatrix}$$

Due to the synchronous scheduling assumption mentioned in Section III.A.1, the data distribution can be made in discrete cycles synchronously. Given an initial possession matrix P^0 , after one cycle of file piece distribution, a new possession matrix P^1 will be formed. That is, P^k denotes the possession matrix after k cycles.

In each cycle, Peer i can send out at most p_i file pieces and receive at most q_i file pieces from others. We use two integer vectors $p = \{p_1, p_2, \dots, p_N\}$ and $q = \{q_1, q_2, \dots, q_N\}$ to denote the upload and download limits for the peers. For example in Fig. 1, $p = \{1, 1, 2, 2, 2\}$ and $q = \{2, 3, 2, 3, 3\}$. Note that it is usual to have p_i smaller than q_i in asymmetric bandwidth allocation schemes, such as ADSL. Nonetheless, our algorithms and analysis presented in later sections are not restricted by whether p_i is smaller or larger than q_i .

We also refer to a possession matrix as a *problem instance* since it provides all the information we need to solve the

problem. A problem instance P is *feasible* if for each file piece in $\{F_1, F_2, \dots, F_M\}$, at least one peer possesses it, i.e. $\sum_{i=1}^N P_{ij} \geq 1, \forall j \in [1, M]$. A problem instance is *infeasible* if it is not feasible, meaning that there is no way for every peer to get all file pieces since there is at least one file piece not available in the system.

A *schedule* specifies how file pieces are distributed among peers. At each cycle, for each peer, it specifies which file pieces the peer has to send out and to whom. A possible schedule for P^0 above with $p = \{1, 1, 2, 2, 2\}$ and $q = \{2, 3, 2, 3, 3\}$ is:

Node 1: send Piece 3 to Node 2

Node 2: send Piece 4 to Node 1

Node 3: send Piece 5 to Node 1, send Piece 5 to Node 2

Node 4: send Piece 6 to Node 2, send Piece 6 to Node 3

Node 5: send Piece 2 to Node 4, send Piece 7 to Node 4

Formally, we use an $N \times M$ matrix S to represent the schedule in one cycle, which specifies which pieces a peer receives and from whom. $S_{ij} = x$ if and only if node i receives file piece j from node x , otherwise $S_{ij} = 0$. From S , we can derive the piece transmission matrix T (also $N \times M$), which ignores the sender identities and only specifies which pieces a peer receives. $T_{ij} = 1$ if and only if $S_{ij} \neq 0$, otherwise $T_{ij} = 0$. Similar to P , we use superscripts to refer to different cycles. That is, S^k and T^k are the schedule and piece transmission matrix at cycle k , respectively. For the above example schedule, we have:

$$S^0 = \begin{pmatrix} 0 & 0 & 0 & 2 & 3 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 3 & 4 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 4 & 0 & 0 \\ 0 & 5 & 0 & 0 & 0 & 0 & 5 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} T^0 = \begin{pmatrix} 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Node 1 receives Piece 4 sent from Node 2 and Piece 5 sent from Node 3. Therefore, $S_{14}^0 = 2, S_{15}^0 = 3$ and $T_{14}^0 = 1, T_{15}^0 = 1$.

There are several properties that a *valid* schedule (a schedule that does not violate any assumptions) should observe and they are listed as follows. (Superscripts in matrices are dropped to enhance readability when the context is clear.)

- At least one file piece must be distributed among the peers in each cycle (at least one entry in S and T is non-zero)
- A node cannot send a piece that it does not possess (if $S_{ij} = x$, then $P_{xj} = 1$)
- Node x cannot send more than p_x file pieces in a cycle (at most p_x x 's in S)
- Node x cannot receive more than q_x file pieces in a cycle (sum of row x in T is at most q_x)
- Node x cannot send a piece to Node y that Node y already has ($S_{yj}^k = 0$ and $T_{yj}^k = 0$ if $P_{yj}^k = 1$ for the same y, j at any cycle k)

Given the possession matrix P^{k-1} and a valid schedule S^{k-1}, T^{k-1} at cycle $k-1$, the possession matrix at cycle k can be obtained by adding P^{k-1} and T^{k-1} . Mathematically, for $k > 0$ and is an integer,

$$P^k = P^{k-1} + T^{k-1}. \quad (1)$$

Referring to the above example,

$$P^1 = P^0 + T^0 = \begin{pmatrix} 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 \end{pmatrix}$$

Intuitively, if we keep on applying a valid schedule in each cycle to a feasible problem instance, all peers will get all the file pieces eventually and the file distribution can terminate. In other words, given an initial feasible P^0 and a sequence of valid schedules, after a certain, say k_0 , cycles, $P^{k_0} = \mathbf{1}, \forall i, j$. k_0 is the time needed for complete distribution of the whole file to all peers. An optimal schedule is a schedule that requires the minimum number of cycles for completion among all possible schedules. Our goal is to develop algorithms that aim at finding optimal schedules.

6) *Pull-based vs. Push-based*: Here we distinguish two models for determining the transmission schedule: pull-based and push-based. In both models, peers have to exchange their file piece possession information periodically. They differ in how to make the decisions of which pieces to send and to whom.

The pull-based model is commonly used in existing applications, such as BT, in which the receiver determines which file pieces he needs from others and subsequently sends request messages to the nodes he chooses. The file source who receives these messages could accept or reject the requests based on some policies, such as available bandwidth and the requestors' contributions. One disadvantage of this model is that there will be many short-length but frequent request messages flowing through the network, taking up network bandwidth and processing time. In addition, for the distributed pull-based model, it may happen that all peers try to request the same file piece from the same source, thus wasting queuing time at the source node (or even getting rejected). In the following example problem instance, all nodes may send requests to Node 1 for Piece 1 since it is the rarest piece (as only one node in the network has it). We refer to this problem as request collision.

$$P = \begin{pmatrix} 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 & 1 \end{pmatrix}$$

In view of the above problem, we propose the push-based model, in which the sender decides which file pieces and to whom he likes to send. In the beginning of a cycle, every node constructs the possession matrix P based on the possession information from its peers and this matrix is the same for every node. Based on P , each peer determines the file pieces to send and the recipients directly, without the need of request messages used in the pull-based approach. To avoid the request collision problem, an algorithm that generates valid schedules is employed to determine what each peer should do based on the possession matrix. As long as each peer executes the same valid scheduling algorithm using the same matrix, peers should send file pieces without any conflict.

In Section IV, we describe several scheduling algorithms that are suitable for the push-based model and evaluate their

performance by simulations in Section V.

B. Analysis

In this section, we analyze the lower bound of k_0 , which is the number of cycles needed for complete distribution of the whole file to all peers. The peers may have different upload and download limits for each synchronous cycle defined by the vectors $\mathbf{p} = \{p_1, p_2, \dots, p_N\}$ and $\mathbf{q} = \{q_1, q_2, \dots, q_N\}$, respectively. We only consider the case that the problem instance is feasible.

1) *Across Rows*: Let r_i be the total number of θ s across row i in \mathbf{P} , i.e. $r_i = \sum_{j=1}^M (1 - P_{ij})$, the minimum value of k_0 is given by

$$k_0 \geq \max_{i \in [1, N]} \left\lceil \frac{r_i}{q_i} \right\rceil. \quad (2)$$

Proof: Peer i requires at least $\lceil r_i/q_i \rceil$ cycles to get the whole file since he can get at most q_i pieces for each cycle and he needs a total of r_i pieces. The node with the maximum value of $\lceil r_i/q_i \rceil$ gives the lower bound of cycles required.

2) *Along Columns*: Let p_{\max} be the maximum value among the upload limit vector \mathbf{p} , i.e. $p_{\max} = \max_{i \in [1, N]} \{p_i\}$. Let c_j be the total number of $\mathbf{1}$ s along column j in \mathbf{P} , i.e. $c_j = \sum_{i=1}^N P_{ij}$, we can find the minimum number of $\mathbf{1}$ s along all columns $c_{\min} = \min_{j \in [1, M]} \{c_j\}$. The minimum value of k_0 is given by

$$k_0 \geq \left\lceil \log_{1+p_{\max}} \left(\frac{N}{c_{\min}} \right) \right\rceil. \quad (3)$$

Proof: For every cycle, we try to best distribute the particular file piece, say Piece x , that the fewest peers have. At Cycle 0, there are c_{\min} $\mathbf{1}$ s along Column x . At Cycle 1, there will be at most $c_{\min}(p_{\max}+1)$ $\mathbf{1}$ s along Column x , since there are c_{\min} nodes that possesses this file piece and each of them can send out at most p_{\max} $\mathbf{1}$ s to others. At Cycle 2, there are at most $c_{\min}(p_{\max}+1)^2$ $\mathbf{1}$ s, and so on. At Cycle k_0 , there will be at most $c_{\min}(p_{\max}+1)^{k_0}$ $\mathbf{1}$ s along Column x , which should be greater than or equal to N . That is, $c_{\min}(p_{\max}+1)^{k_0} \geq N$. As k_0 must be an integer, it gives $k_0 \geq \left\lceil \log_{1+p_{\max}} \left(\frac{N}{c_{\min}} \right) \right\rceil$.

3) *Whole Matrix*: Let z be the total number of θ s in \mathbf{P} , i.e. $z = \sum_{i=1}^N \sum_{j=1}^M (1 - P_{ij})$. Let p_{sum} be the sum of p_i in \mathbf{p} , i.e. $p_{\text{sum}} = \sum_{i=1}^N p_i$, and q_{sum} be the sum of q_i in \mathbf{q} , i.e. $q_{\text{sum}} = \sum_{i=1}^N q_i$. The minimum value of k_0 is given by

$$k_0 \geq \left\lceil \frac{z}{\min\{p_{\text{sum}}, q_{\text{sum}}\}} \right\rceil. \quad (4)$$

Proof: The maximum number of file pieces that can be exchanged in one cycle is $\min\{p_{\text{sum}}, q_{\text{sum}}\}$, and there are a total of z un-received pieces in the network, thus at least $\left\lceil \frac{z}{\min\{p_{\text{sum}}, q_{\text{sum}}\}} \right\rceil$ cycles are required for complete distribution.

4) *Lower Bound*: Combining (2), (3), (4), the lower bound of the value of k_0 is the maximum value of the three, i.e.

$$k_0 \geq \max \left\{ \max_{i \in [1, N]} \left\lceil \frac{r_i}{q_i} \right\rceil, \left\lceil \log_{1+p_{\max}} \left(\frac{N}{c_{\min}} \right) \right\rceil, \left\lceil \frac{z}{\min\{p_{\text{sum}}, q_{\text{sum}}\}} \right\rceil \right\}. \quad (5)$$

In the following example, $k_0 \geq \max\{0, 3, 2, 3, 2, 3\} = 3$ (by (2)),

$k_0 \geq \left\lceil \log_{1+2} \left(\frac{6}{1} \right) \right\rceil = 2$ (by (3)), and $k_0 \geq \left\lceil \frac{33}{\min\{9, 19\}} \right\rceil = 4$ (by (4)). Thus, at least four cycles are needed for complete distribution (by (5)).

$$\mathbf{P} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{pmatrix} \quad \mathbf{p} = \begin{pmatrix} 2 \\ 2 \\ 1 \\ 1 \\ 1 \end{pmatrix} \quad \mathbf{q} = \begin{pmatrix} 3 \\ 3 \\ 4 \\ 3 \\ 4 \\ 2 \end{pmatrix}$$

From simulations, we find (5) can return the optimal number of cycles for most cases we tested and we shall use the value estimated by (5) for performance comparison in later sections.

IV. SCHEDULING ALGORITHMS

We now present three types of transmission scheduling algorithms. They are *Rarest Piece First (RPF)*, *Most Demanding Node First (MDNF)*, and *Maximum-Flow (MaxFlow)* algorithms. All of them run in polynomial time.

A. Rarest Piece First (RPF)

The *Rarest Piece First* algorithm is borrowed from the *Rarest Element First* algorithm used in BitTorrent. In *RPF*, those file pieces that most peers do not have (rarest) are distributed first.

Definition 1: The *rarity* c_j of Piece j is the number of peers who have Piece j . That is, $c_j = \sum_{i=1}^N P_{ij}$.

RPF aims at increasing the availability of different file pieces in the network, to maximize the chance that peers may still have some pieces that other peers want. In case the file is published by a single source who may just seed (remain available to contribute) the file for a short period of time, *RPF* also tries to distribute all pieces from the original source to different peers across the network as quickly as possible, so that the distribution can continue even if the original source leaves.

In *RPF*, each peer chooses the rarest piece (with smallest c_j) to send out among those pieces he currently has. There is no preference on the choice of recipients; this piece is sent to those with lower row indices who do not have this piece, have not been assigned to receive this piece from others and have not exceeded their download limits. If after distributing this piece to all possible recipients, the peer still has remaining upload bandwidth, he will try to distribute the next rarest piece until he has no more upload bandwidth or there are no more possible recipients. This process is done node by node (i.e. row by row in the possession matrix). For each cycle, the complexity of this algorithm is $O(NM(N+\log M))$, where N is the number of peers and M is the number of file pieces. For example, with $\mathbf{p} = \{1, 1, 2, 2, 2\}$ and $\mathbf{q} = \{2, 3, 2, 3, 3\}$,

$$P^0 = \begin{pmatrix} 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} P^1 = \begin{pmatrix} 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 \end{pmatrix}$$

The schedule determined for P^0 is (arrows are added for ease of understanding):

- Node 1: send Piece 3 to Node 2
- Node 2: send Piece 4 to Node 1
- Node 3: send Piece 5 to Node 1, send Piece 5 to Node 2
- Node 4: send Piece 6 to Node 2, send Piece 6 to Node 3
- Node 5: send Piece 2 to Node 4, send Piece 7 to Node 4

At P^0 , Node 1 is the first to make decision and it chooses Piece 3 because among Pieces 1 and 3 it currently has, Piece 3 is the rarest (only two peers have it, while three peers have Piece 1). Then it will be sent to Node 2, since Node 2 does not have Piece 3 and has available download bandwidth. Similarly, Node 2 chooses its rarest Piece 4 to send to Node 1. Node 3 can send out two pieces for each cycle. It chooses its rarest Piece 5 and selects Nodes 1 and 2 as recipients. Node 4 sends Piece 6 to Nodes 2 and 3 (not Node 1 since Node 1 has been assigned to receive $q_1 = 2$ pieces already). Node 5 first tries to send out Piece 2, and Node 4 is the only possible recipient as Nodes 1 and 2 have already reached their download limits. Node 5 can send out one piece more, so it sends the next rarest Piece 7 to Node 4 also. The resulting problem matrix at the next cycle, P^1 , is also shown with those 1 s corresponding to the pieces just transmitted underlined.

However, the performance of *RPF*, which is derived from the *Rarest Element First* algorithm of BitTorrent, is unsatisfactory as shown by simulation, and it performs much worse than the following algorithms.

B. Most Demanding Node First (MDNF)

As indicated in Section III.B, the number of cycles needed depends on two factors: how many pieces a peer needs and how rare a file piece is. To reduce the time for distribution, both factors have to be considered. *RPF* only considers the second factor, while the *Most Demanding Node First* algorithm takes care of the first factor by adding one additional criterion for choosing recipients and the performance improvement over *RPF* is significant with this simple enhancement.

Definition 2: The demand d_i of Node i is the number of un-received pieces for Node i . That is, $d_i = \sum_{j=1}^M (1 - P_{ij})$.

We attach a demand d_i to every node and we prefer to send to the node with largest d_i . In case several nodes have the same demand, we just send to the node with the lowest row index.

Similar to *RPF*, we schedule the transmissions node by node; while choosing recipients we prefer the one with the highest demand d_i that does not have the piece and has not exceeded its download limit yet. Its complexity is the same as *RPF*, i.e. $O(NM(N + \log M))$. For example, with $p = \{1, 1, 2, 2, 2\}$ and $q = \{2, 3, 2, 3, 3\}$,

$$P^0 = \begin{pmatrix} 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} P^1 = \begin{pmatrix} 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 \end{pmatrix}$$

The schedule is determined in this sequence:

- Node 1: send Piece 3 to Node 2
- Node 2: send Piece 4 to Node 1
- Node 3: send Piece 5 to Node 1, send Piece 5 to Node 2
- Node 4: send Piece 6 to Node 2, send Piece 6 to Node 5
- Node 5: send Piece 2 to Node 4, send Piece 7 to Node 4

The demands for each node are written at the right of P^0 . At P^0 , Node 1 chooses its rarest piece, Piece 3, to send out and chooses the most demanding node, Node 2, to receive this piece. It is similar for Nodes 2 and 3. Node 4 now sends Piece 6 to Nodes 2 and 5 (instead of Node 3) because Node 5 is more demanding than Node 3, and so on for other nodes.

MDNF performs better than *RPF* in most cases but is still not the best. A common characteristic that is shared between *RPF* and *MDNF* is that the maximum number of transmissions for each cycle cannot be achieved. In the following simple example, $p = \{2, 2, 1, 1\}$ and $q = \{2, 2, 2, 2\}$. Using *MDNF*, only five transmissions can be scheduled (but the maximum is six).

$$P^0 = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{pmatrix} \begin{matrix} 2 \\ 2 \\ 2 \\ 2 \end{matrix}$$

To fix the problem, we find as many transmissions as possible in each cycle and the algorithm is described in the next section.

C. Maximum-Flow Algorithms (MaxFlow)

In this section, we present a novel maximum-flow graph model for finding transmission schedules which outperforms the above two algorithms. We transform the problem instance to the well-known *maximum-flow problem* so as to find the maximum number of transmissions in each cycle. Weights are added to the nodes to achieve better matching. We first describe how to transform the problem and then explain the algorithm in detail.

1) *Flow Network*: A flow network graph $G=(V,E)$ is a directed graph in which each edge $(u,v) \in E$ has a nonnegative capacity $c(u,v) \geq 0$. There are two special vertices, the source s and the sink t . A flow f in G is a real-valued function $f:V \times V \rightarrow \mathbb{R}$ that satisfies the following three properties:

- Capacity Constraint: the flow from one vertex to another must not exceed the given capacity, i.e. $f(u,v) \leq c(u,v) \quad \forall u,v \in V$.
- Skew Symmetry: the flow from a vertex u to a vertex v is the negative of the flow in the reverse direction, i.e. $f(u,v) = -f(v,u) \quad \forall u,v \in V$.
- Flow Conservation: the total flow out of a vertex other than the source or sink is zero, i.e. $\sum_{u \in V} f(u,v) = 0 \quad \forall v \in V - \{s,t\}$.

The value of a flow f is defined as $|f| = \sum_{v \in V} f(s,v)$, i.e. the total flow out of the source. In the *maximum-flow problem*, we wish

to find a flow of maximum value. Due to space limitation, we refer interested readers to [19] for more formal discussions.

2) *Problem Transformation*: We now describe how to construct the flow network graph from a problem instance P .

Definition 3: The flow network graph from P is a directed graph $G=(V,E)$. The vertices V can be separated into four groups, i.e., $V=L \cup R \cup B \cup \{s,t\}$. $L=\{L_1, L_2, \dots, L_N\}$ refers to the N peers acting as senders in the file sharing session and $R=\{R_1, R_2, \dots, R_N\}$ refers to the same N peers acting as receivers. L and R have the same cardinality as the number of peers N , i.e. $|L|=|R|=N$. For each node in R , we create an *un-received piece node* for each un-received file piece it demands and put these nodes into the set B , i.e. $b_{ij} \in B \Leftrightarrow P_{ij}=0$. It has the same cardinality as the total number of 0 s in P , i.e. $|B|=\sum_{i=1}^N \sum_{j=1}^M (1-P_{ij})$. s, t are the source and sink nodes, respectively. The edges, $E=\{(s, L_i) | L_i \in L\} \cup \{(R_i, t) | R_i \in R\} \cup \{(b_{ij}, R_i) | b_{ij} \in B, R_i \in R\} \cup H$, also consist of four sets. $\{(s, L_i) | L_i \in L\}$, $\{(R_i, t) | R_i \in R\}$ are the sets of edges from the source s to left-side nodes L and right-side nodes R to the sink t , respectively. $\{(b_{ij}, R_i) | b_{ij} \in B, R_i \in R\}$ are the edges from the un-received piece nodes for node i to the right-side node R_i . $H=\{(L_u, b_{vj}) | L_u \in L, b_{vj} \in B, \text{and } (P_{uj}=1 \wedge P_{vj}=0)\}$ are the edges from left-side nodes L to the un-received piece nodes B and depend on the possession matrix P . There is an edge from L_u to b_{vj} if Peer u has file piece j , while Peer v does not have it, so that Peer u can send Piece j to Peer v . The edge capacities are integer-valued, $c(s, L_i)=p_i$ and $c(R_i, t)=q_i$, where p_i and q_i are the upload and download limits for Peer i defined by the vectors $p=\{p_1, p_2, \dots, p_N\}$ and $q=\{q_1, q_2, \dots, q_N\}$. $c(s, L_i)$ dictates that Node i sends at most p_i pieces out. Similarly $c(R_i, t)$ dictates that Node i receives at most q_i pieces from others. $c(u, v)=1 \quad \forall (u, v) \in E \setminus (\{(s, L_i)\} \cup \{(R_i, t)\})$ for other edges excluding those from source s to L_i and those from R_i to sink t .

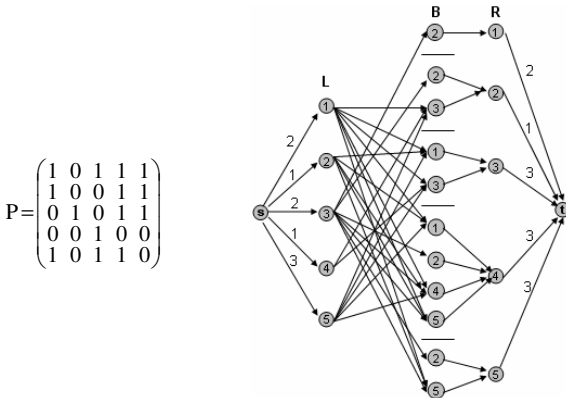


Fig. 2. Possession matrix and its flow network graph

Fig. 2 illustrates the transformation process. For this possession matrix P , with $p=\{2, 1, 2, 1, 3\}$ and $q=\{2, 1, 3, 3, 3\}$, the flow network is shown beside P . There are a total of eleven

un-received piece nodes in B , as there are eleven 0 s in P . b_{12} corresponds to the un-received Piece 2 for Peer 1, b_{22} corresponds to Piece 2 for Node 2, etc. There are links from L_1 to $b_{23}, b_{31}, b_{44}, b_{55}$, etc., but not to b_{22}, b_{42}, b_{52} , etc., since Peer 1 can send Piece 3 to Peer 2, Piece 1 to Peer 3, Piece 4 to Peer 4, Piece 5 to Peer 5, but cannot send Piece 2 to other peers as Peer 1 does not have Piece 2. The arguments for other links in H are similar. The capacities for the edges from s to L_i and R_i to t are written on top of the corresponding edges and follow the given upload and download limit vectors p and q . There are $O(N^2M)$ edges and the complexity for constructing the flow network graph from P is also $O(N^2M)$.

To find the maximum-flow $|f^*|$, we adopt the well-established *Edmonds-Karp* algorithm, which is a particular implementation of the general *Ford-Fulkerson* method [19]. It finds augmenting paths by using breath-first search from the source s to the sink t . Nodes with smaller indices have higher preference in expanding states in breath-first search. Its complexity is $O(|E||f^*|)=O\left(N^2M \times \min\left\{\sum_{i=1}^N p_i, \sum_{i=1}^N q_i\right\}\right)$, which is still linear in M and N , and is worth the implementation effort. In fact, the small increase in scheduling complexity would not add much to the overall transmission time as the computation time is very small compared with the actual data transmission time of file pieces.

For the example problem instance in Fig. 2, we will have a maximum-flow $|f^*|=9$ as in Fig. 3, with matched flows highlighted. An example translation of a matched flow to a schedule is: $s \rightarrow L_1 \rightarrow B_{45} \rightarrow R_4 \rightarrow t$ means that Node 1 sends Piece 5 to Node 4. The transmission schedule is thus:

- Node 1: send Piece 5 to Node 4, send Piece 5 to Node 5
- Node 2: send Piece 1 to Node 4
- Node 3: send Piece 2 to Node 1, send Piece 2 to Node 5
- Node 4: send Piece 3 to Node 3
- Node 5: send Piece 3 to Node 2, send Piece 1 to Node 3, send Piece 4 to Node 4

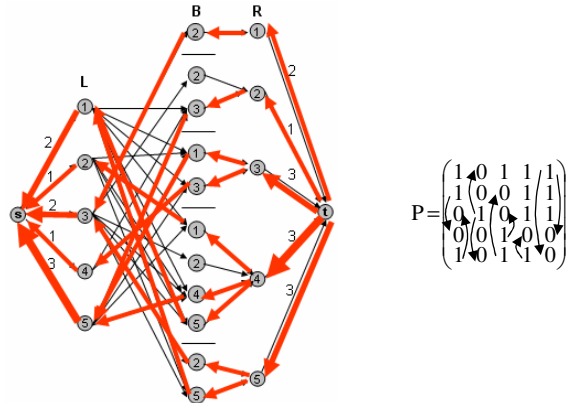


Fig. 3. Maximum-flow and the scheduled transmissions

3) *MaxFlow – Weighted*: Although *MaxFlow* always returns a schedule with the maximum number of transmissions for each cycle, the performance is unsatisfactory, as it does not consider whether we can match more in subsequent cycles. It does not take care of the rarity of file pieces and the demands of

nodes as in *RPF* and *MDNF*. To find a better matching, we put weights on the nodes in L and B , so as to give priorities to some nodes during the matching process. Definitions 4 and 5 define how to measure the rarity of the file pieces a peer possesses and the rarity of the file pieces a peer demands, respectively.

Definition 4: The *rarity possession index* γ_i of Peer i is the sum of number of 0 s in other peers for those pieces that Peer i has. That is, $\gamma_i = \sum_{a=1}^N \sum_{b=1}^M (A_{ab}(i))$ where A_{ab} is an $N \times M$ matrix and

$$A_{ab}(i) = \begin{cases} 1 & \text{if } (a \neq i) \wedge (P_{ib} = 1) \wedge (P_{ab} = 0). \\ 0 & \text{otherwise} \end{cases}$$

Definition 5: The *rarity demand index* δ_{ij} for file piece j that Peer i demands is the sum of number of 0 s across Row i and Column j . That is, $\delta_{ij} = \sum_{g=1}^M (I - P_{ig}) + \sum_{h=1}^N (I - P_{hj}) - 1$.

Our *MaxFlow – Weighted* algorithm works as follows:

1. Construct the flow network graph from the possession matrix P according to Definition 3.
2. Find the *rarity possession indices* γ_i of all peers and assign the values as weights on the nodes in L accordingly.
3. Find the *rarity demand indices* δ_{ij} of all missing pieces of all peers and assign the values as weights on the nodes in B accordingly.
4. Employ the *Edmonds-Karp* algorithm, with preference in expanding states with higher weights to find the weighted maximum flow.
5. From the maximum flow returned, identify the pieces and recipients that each peer needs to send to.

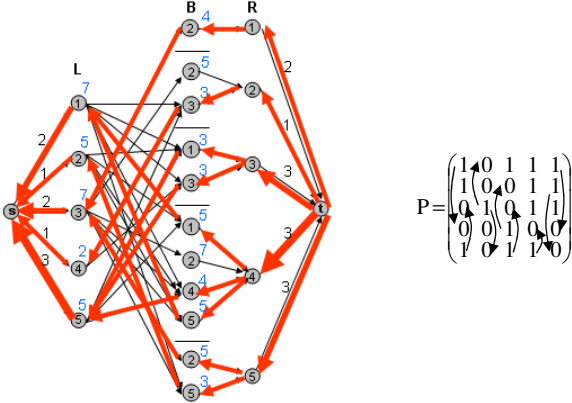


Fig. 4. *MaxFlow – Weighted* graph and its scheduled transmissions

Consider an example in Fig. 4, with $p = \{2, 1, 2, 1, 3\}$ and $q = \{2, 1, 3, 3, 3\}$. The numbers beside the nodes in L are the rarity possession indices γ_i , and those beside the un-received piece nodes B are the rarity demand indices δ_{ij} . For example, L_3 has $\gamma_3 = 4 + 1 + 2 = 7$ because for Pieces 2, 4, 5 that Peer 3 has, there are a total of seven 0 s along Columns 2, 4, and 5. B_{42} has $\delta_{42} = 7$ because there are a total of seven 0 s across Row 4 and along Column 2. By preferring paths with the largest γ_i first, we ensure those peers who have rare pieces can send first. On the other hand, by preferring paths with largest δ_{ij} first, we tend to select a node with higher demand to receive a rare piece first.

Simulation results show that *MaxFlow – Weighted* performs

better than *MDNF* and *RPF*. However, there are still a few cases that it cannot achieve the optimal as shown below, with $p = \{2, 2, 2, 2\}$ and $q = \{3, 3, 3, 3, 3\}$.

Using *MaxFlow – Weighted*, total 3 cycles are needed.

$$P^0 = \begin{pmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 \end{pmatrix} \begin{matrix} 4 \\ 4 \\ 2 \\ 5 \\ 5 \\ 5 \\ 4 \\ 4 \end{matrix} \quad P^1 = \begin{pmatrix} 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 \end{matrix} \dots P^3 = 1$$

Using *MDNF*, only 2 cycles are needed.

$$P^0 = \begin{pmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 \end{pmatrix} \begin{matrix} 4 \\ 4 \\ 2 \\ 5 \\ 5 \\ 5 \\ 4 \\ 4 \end{matrix} \quad P^1 = \begin{pmatrix} 1 & 1 & 1 & 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \end{matrix} P^2 = 1$$

The sums of 0 s across rows and columns are written beside P^0 . The key point is that at P^0 , the rarity demand indices δ_{ij} are similar in value. Peer 8 gets nothing at P^0 since his δ_{8j} are mostly smaller than or equal to others and he has the lowest row index. Note that the weights are kept constant for the whole duration of each cycle and will not be changed even if some partial assignments have been made. Thus, when choosing recipients those peers who have been assigned to receive something may still be further assigned to receive more as their static weights remain as the highest, resulting in unfair resource allocation.

4) *MaxFlow – Dynamically-Weighted*: Due to the above problem, we further enhance *MaxFlow – Weighted* by allowing the weights on nodes to be dynamically varied within each scheduling cycle. Whenever Peer i is temporarily assigned to receive Piece j during the scheduling process, the rarity possession indices γ_i and rarity demand indices δ_{ij} will be changed and hence the weights on nodes will also be changed. Then the scheduling process continues using the new weights.

Consider the above counter example in Section IV.C.3, we can derive γ_i and δ_{ij} from the sums of 0 s written beside the rows and columns, e.g. $\gamma = \{15, 14, 25, 13, 15, 10, 16, 16\}$ and $\delta_{43} = 9$ which is the greatest value among all δ_{ij}). After first assigning Peer 3 to send Piece 3 to Peer 4, P_{43} is assigned to 1 temporarily, the sums of 0 s will be changed as shown below and hence the weights also be changed. The next scheduling step will follow these new weights to compute the next assignment. Then this P^0 can be completed in two cycles, which is the optimal.

$$P^0 = \begin{pmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 \end{pmatrix} \begin{matrix} 4 \\ 4 \\ 2 \\ 5 \\ 4 \\ 5 \\ 4 \\ 4 \end{matrix} \quad \xrightarrow{\text{After 1st assignment}} \quad P^0 = \begin{pmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 \end{pmatrix} \begin{matrix} 4 \\ 4 \\ 2 \\ 5 \\ 4 \\ 5 \\ 4 \\ 4 \end{matrix}$$

From simulations, *MaxFlow – Dynamically-Weighted* performs the best among all algorithms presented and it can achieve the optimal for most cases tested. It is the best algorithm we have developed so far.

V. SIMULATION RESULTS

We randomly generate the problem instances (with each individual matrix element independently generated) and employ various algorithms presented in the previous section to find transmission schedules. We simulated 1000 cases for each simulation setting. We note the average number of cycles needed for complete distribution by using that particular algorithm and compare it with the lower bound (5).

A. Equal probability of 1s and 0s in P , $p_i = 2$, $q_i = 3$

For each matrix element in P , the probability of generating a 1 or 0 is the same, i.e. 0.5. All peers have the same upload and download limits of $p_i = 2$ and $q_i = 3$ for each cycle.

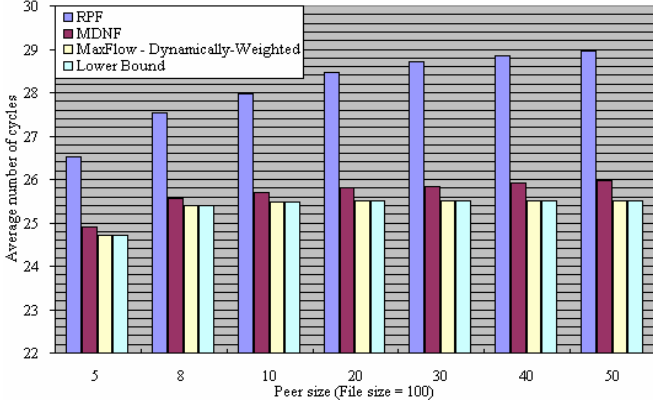


Fig. 5. Performance comparison of representative scheduling algorithms with varying peer sizes (file size = 100, $p_i = 2$, $q_i = 3$, equal probability for 1s and 0s)

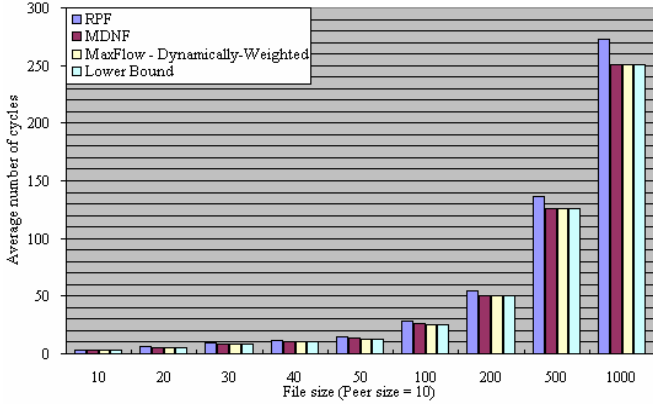


Fig. 6. Performance comparison of representative scheduling algorithms with varying file sizes (peer size = 10, $p_i = 2$, $q_i = 3$, equal probability for 1s and 0s)

Fig. 5 and Fig. 6 show the performance comparison of the three representative algorithms with varying peer sizes and file sizes respectively. In all cases, the average number of cycles used by *MaxFlow – Dynamically-Weighted* is smaller than that by *MDNF*, which is in turn smaller than that by *RPF*. Also, *MaxFlow – Dynamically-Weighted* can achieve the same value as the lower bound in all cases, which implies that it already achieves the optimum for all cases tested in these graphs. For varying peer sizes as shown in Fig. 5, the average number of cycles used remains more or less constant with increasing peer size. This illustrates the good scalability of P2P file sharing approach; though more peers are requesting the same file, they

also contribute their outgoing bandwidth for sharing. For varying file sizes as shown in Fig. 6, the performance improvement of *MaxFlow – Dynamically-Weighted* over *RPF* and *MDNF* actually increases with increasing file size (though this cannot be easily seen due to the graph scale). In existing P2P file sharing networks, such as BT, the number of simultaneously connected peers is kept at about 10~30, while the number of file pieces is about 2000~4000 for a typical file of 500MBytes~1GByte with file piece size of 256KBytes. The performance improvement of *MaxFlow – Dynamically-Weighted* over other algorithms becomes more significant with this large number of file pieces.

B. Equal probability of 1s and 0s in P , $p_i = 3$, $q_i = 7$

We try different values of p_i and q_i to see if there is any significant effect on the performance. We try $p_i = 3$ and $q_i = 7$, which is closer to the actual bandwidth allocation in ADSL. The results are similar to Fig. 5 and 6, demonstrating that variations in p_i and q_i will not have significant effects on the performance.

C. Probability of 1s : 0s = 1 : 2, $p_i = 2$, $q_i = 3$

We try to have different probabilities for generating a 1 and a 0 for each matrix element in P . We choose the ratio of the probability of generating a 1 to that of a 0 to be 1:2, meaning that the network has fewer file pieces at the beginning, corresponding to an earlier stage of the file sharing session. The results are shown in Fig. 7 and Fig. 8.

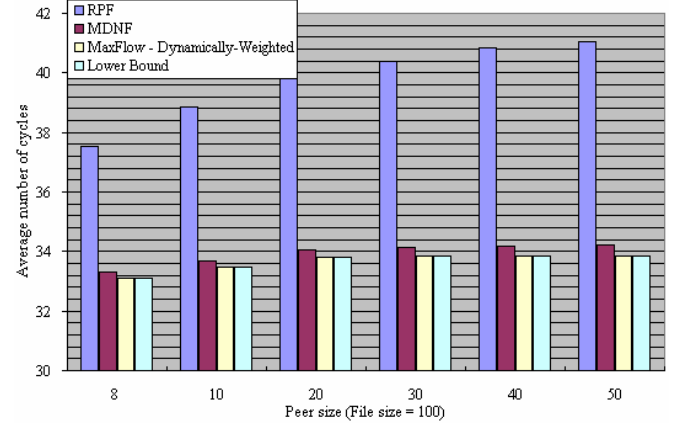


Fig. 7. Performance comparison of representative scheduling algorithms with varying peer sizes (file size = 100, $p_i = 2$, $q_i = 3$, probability of 1s:0s = 1:2)

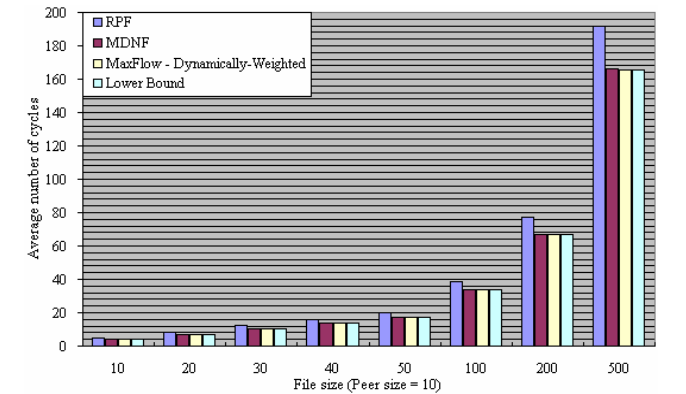


Fig. 8. Performance comparison of representative scheduling algorithms with varying file sizes (peer size = 10, $p_i = 2$, $q_i = 3$, probability of 1s:0s = 1:2)

From Fig. 7 and Fig. 8, when the network has fewer file pieces at the beginning, the performance improvement of *MaxFlow – Dynamically-Weighted* over *RPF* is greater than that when the network has more available file pieces. As in Fig. 7, when the peer size is 50 and the file size is 100, the percentage improvement of *MaxFlow – Dynamically-Weighted* over *RPF* is 21.28%. For the same problem size in Fig. 5, the percentage improvement is only 13.56%. Other observations are basically the same as that in Section V.A.

VI. DISCUSSION AND FUTURE WORK

In this paper, we have investigated the simplified problem of P2P file distribution scheduling with asymmetric bandwidth allocation and synchronous scheduling assumptions. This scenario may appear in some private networks of a large company where some critical content has to be quickly replicated on a large number of machines. All the peers are known beforehand and they have similar connectivity and bandwidth capacities and the distribution process stops once the content has been fully replicated. Obviously, such static and synchronous scenarios are rare in real-world systems, but this preliminary study provides insights on the peer selection and file piece selection strategies, and evaluates their performance.

In our future research, we shall study the case of asynchronous scheduling in heterogeneous networks, in which the transmission time for sending a message between different pairs of nodes is different. We shall also study the case when the network is dynamic, in which the peers may come and go at will and they may shift to communicate with different sets of peers during the file distribution process.

VII. CONCLUSION

Peer-to-Peer file-sharing applications have become immensely popular in the Internet, but previous research seldom investigates the data distribution problem which should be the core of any file sharing applications. We formally define the collaborative file distribution problem with the possession and transmission matrix formulation and deduce a theoretical bound for the minimum distribution time required. We develop several types of algorithms (*RPF*, *MDNF* and *MaxFlow*) for solving the scheduling problem of deciding who to send which file pieces, and to whom. In particular, our novel dynamically weighted maximum-flow algorithm outperforms other algorithms and can return the optimal solution for most cases as shown by simulations. Therefore, we conclude that the *MaxFlow – Dynamically-Weighted* algorithm is a promising algorithm for practical deployment in P2P file sharing applications.

REFERENCES

[1] The official BitTorrent website, <http://www.bittorrent.com/>
 [2] The official Gnutella website, <http://www.gnutella.com/>
 [3] The official Kazaa website, <http://www.kazaa.com/>

[4] The official Napster website, <http://www.napster.com/>
 [5] A. Parker, "The true picture of peer-to-peer file-sharing," *CacheLogic Presentation*, July 2004
 [6] P. Rodriguez, and E.W. Biersack, "Dynamic parallel access to replicated content in the Internet," *IEEE/ACM Transactions on Networking*, Vol. 10, No. 4, pp. 455-465, August 2002
 [7] X. Yang and G. de Veciana, "Service capacity of peer to peer networks," *Proc. IEEE INFOCOM 2004*, Hong Kong, China, March 2004
 [8] B. Cohen, "Incentive build robustness in BitTorrent," <http://www.bittorrent.com/bittorrentecon.pdf>, May 2003
 [9] E.W. Biersack, P. Rodriguez, and P. Felber, "Performance analysis of peer-to-peer networks for file distribution," *Proc. 5th Int'l Workshop on Quality of Future Internet Services*, pp. 1-10, September 2004
 [10] P. Felber, and E.W. Biersack, "Cooperative content distribution: scalability through self-organization," *Self-star Properties in Complex Information Systems*, O. Babaoglu et al., pp. 343-357, Springer-Verlag, Berlin Heidelberg, 2005
 [11] P.B. Bhat, C.S. Raghavendra, and V.K. Prasanna, "Efficient collective communication in distributed heterogeneous systems," *Proc. Int'l Conf. on Distributed Computing Systems*, pp. 15-24, June 1999
 [12] S. Khuller, and Y.A. Kim, "On broadcasting in heterogeneous networks," *Proc. ACM-SIAM Symposium on Discrete Algorithms*, pp. 1011-1020, January 2004
 [13] M. Banikazemi, V. Moorthy, and D.K. Panda, "Efficient collective communication on heterogeneous networks of workstations," *Int'l Conf. on Parallel Processing*, pp. 460-467, August 1998
 [14] P. Liu, "Broadcast scheduling optimization for heterogeneous cluster systems," *Journal of Algorithms*, Vol. 42, No. 1, pp. 135-152, January 2002
 [15] T.S.E. Ng, Y.H. Chu, S.G. Rao, K. Sripanidkulchai, and H. Zhang, "Measurement-based optimization techniques for bandwidth-demanding peer-to-peer systems," *Proc. IEEE INFOCOM 2003*, San Francisco, CA, USA, April 2003
 [16] J. Pouwelse, P. Garbacki, D. Epema, and H. Sips, "The BitTorrent P2P file-sharing system: measurements and analysis," *Proc. Int'l Workshop on Peer-to-Peer Systems*, Ithaca, NY, USA, February 2005
 [17] D. Qiu, and R. Srikant, "Modeling and performance analysis of BitTorrent-like peer-to-peer networks," *Proc. ACM SIGCOMM 2004*, Portland, OR, USA, August 2004
 [18] J.S.K. Chan, V.O.K. Li, and K.S. Lui, "Scheduling algorithms for peer-to-peer collaborative file distribution," *Int'l Conf. on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom)*, San Jose, CA, USA, December 2005
 [19] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein, *Introduction to Algorithms*, MIT Press, Cambridge, MA, 2nd edition, 2001