A Global Contribution Approach to Maintain Fairness in P2P Networks

Hiroshi Nishida, *Member*, *IEEE*, and Thinh Nguyen, *Member*, *IEEE*

Abstract—For many P2P systems, implementing right incentives and policies to promote efficient and fair resource sharing is the key to improve the overall system performance. In this paper, we propose a points-based incentive mechanism named Global Contribution (GC) approach that efficiently and naturally maintains fairness in a P2P network. In this approach, a proposed GC algorithm first calculates a global score for each peer that accurately reflects its bandwidth contribution to the entire network. Then, these scores are used in a proposed data transfer policy to determine whether one peer can download data from other peers. Thus, the GC approach achieves: 1) efficiently preventing free-riding, 2) naturally balancing the upload and download amounts in each peer, and 3) reducing rejections in transactions between cooperative peers. Moreover, the GC algorithm requires only private transaction history as an input and can be fully decentralized. Also, its time complexities are approximately $O(N^2)$ in a centralized system and $O(N)$ per peer in a decentralized system.

Index Terms—P2P, free-riding, global contribution, fairness, distributed systems.

1 Introduction

In this paper, we are focused on designing high-level policies that provide right incentives to promote resource sharing in a fair and efficient manner in a P2P network. One of the key factors to promote resource sharing is fairness. A peer in a network feels fair if given the amount of data that it has contributed to other peers, it should be able to download an equal amount. Preventing free-riding is also essential to maintaining fairness in this sense. Another key factor to promote resource sharing is efficiency. Fewer rejections in transactions between cooperative peers will bring higher efficiency to an entire network. In order to achieve them, we need to implement a proper policy which also considers peers' "self-interesting" behavior.

To keep fairness, the popular tit-for-tat policy has been introduced. In tit-for-tat, the transactions between two peers are made according to a policy that "If you give me, I will give you. If you don't give me, I won't give you." In practice, a peer is allowed to download some initial amount of data from another peer that it has never transacted with. This is necessary to bootstrap the sharing process. Otherwise, no peer would be able to share data with anyone since initially a peer has no transaction history with any peer. Though this policy helps keep fairness to some extent, free-riding continues as long as there are peers with which free-riders have never transacted. This is because of the initial amount a free-rider can download. The larger a P2P network is, the more remarkable this problem becomes since a free-rider can find a new peer that it never transacts with, to download its desired data.

Another example of issues regarding fairness currently not addressed by tit-for-tat is *asymmetry of transactions*. In Fig. 1, B downloaded 50 GB from A, C downloaded 50 GB from B, and A downloaded 30 GB from C. Then, A wants to download another 20 GB from C. With tit-for-tat, A might be rejected by C after it downloads some initial amount. This is because A has never uploaded to C. However, this clearly is not fair as A has contributed much data to the system.

This problem is indeed solvable using shared history and maxflow [10]. Using the shared history, every peer keeps all transaction histories that occur in a network. By calculating the maxflow from a downloading peer to an uploading peer using the shared history, the uploading peer can estimate the contribution of the downloading peer. Based on the contribution level, the uploading peer can then decide whether it should share its data with the downloading peer. However, the difficulty with this approach is scalability. The calculation of maxflow takes $O(N^3)$. The shared history is also not feasible both from the viewpoints of storage and bandwidth needed to store and disseminate all transactions to every peer.

We think the fundamental difficulty with many existing approaches is attributed to its relying only on the relationship of literally "peer-to-peer." Instead, if we examine a P2P network globally and determine the contribution of each peer to the entire network, we will be able to use the contribution as "points" to receive benefit from the network. Then, each peer in the network is assigned a point/score or *global contribution* (GC), and we can determine if a transaction should be made between peers according to it. In this paper, we propose: 1) a definition of GC of each peer, 2) distributed and sequential algorithms for computing the GCs, and 3) a protocol for deciding whether a transaction

- *H. Nishida is with the School of EECS, Oregon State University, 1148 Kelley Engineering Center, Corvallis, OR 97331-5501, and ASUSA Corporation, Suite 160, 530 Center St. NE, Salem, OR 97301. E-mail: nishidah@onid.orst.edu.*
- *T. Nguyen is with the School of EECS, Oregon State University, 1148 Kelley Engineering Center, Oregon State University, Corvallis, OR 97331-5501. E-mail: thinhq@eecs.oregonstate.edu.*

Manuscript received 11 May 2009; accepted 21 July 2009; published online 28 July 2009.
Recommended for acceptance by A. Boukerche.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number TPDS-2009-05-0208.
Digital Object Identifier no. 10.1109/TPDS.2009.122.

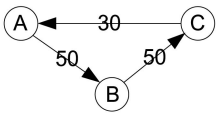

Fig. 1. Example of asymmetry of transactions: B downloaded 50 GB from A, C downloaded 50 GB from B, and A downloaded 30 GB from C. Then, A wants to download another 20 GB from C. With tit-for-tat, A will be rejected.

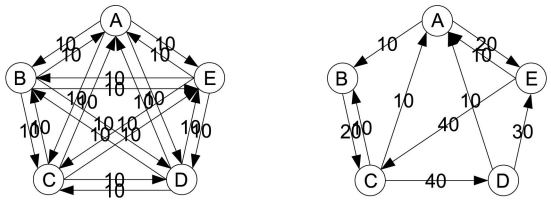

Fig. 2. Examples of the fairest P2P networks: in every node, the total amount of upload = the total amount of download.

should take place between two peers based on their GCs. Then, we show its advantages in our simulation.

The rest of this paper is organized as follows: In Section 2, we provide a brief overview of related work. Section 3 discusses designing criteria for the GC in regard to fairness. Section 4 defines the GC and describes the algorithms for calculating it; both centralized and distributed algorithms are presented. The algorithm is verified in Section 5, then transaction procedures using the algorithm are discussed in Section 6. Section 7 describes some considerations for practical usage. Section 8 presents simulation results for some network models including convergence properties, free-riding prevention, and fairness. Finally, we conclude in Section 9.

2 Related Work

Our algorithm is similar to those of PageRank [25] and EigenTrust [15]. The PageRank rates the popularity of web pages based on the "relative importance of web pages," that is, if a web page is linked from another highly ranked page which has few outlinks, its ranking also becomes high. Thus, the PageRank constructs a system of linear equations (xs are web pages' rankings) according to web links, and obtains each page's ranking by solving the system of linear equations. The biggest difference from the GC is that the PageRank focuses on "popularity" of web pages, not "contribution." As a result, the contribution cannot be expressed in the PageRank's way. In Section 4.4, we compare the PageRank and the GC, and more details are described.

The EigenTrust similarly determines global reputation of each peer. If a peer is relied on by another highly reliable peer, its reputation also becomes high. It also constructs a system of linear equations and obtains reputation of each peer by solving it. The significant difference from the GC is that the EigenTrust does not handle *minus* reliability. To compute "contribution," we have to consider both *plus* contribution that appears as upload and *minus* contribution as download. As a result, the EigenTrust cannot express the contribution. See Section 4.4 for more details.

Many incentive mechanisms have been proposed to cope with unfairness, free-riding, collusion, and whitewashing (peers leave and join with new identities to avoid reputational penalties [11]) in P2P networks [6], [10], [12], [13]. Multilevel Tit-for-tat (ML-TFT) [17], one of the incentive mechanisms recently introduced, ranks peer j from i's perspective as: $M_{i,j} = (i\text{'s download amount from } j)/(i\text{'s total download amount })$. It then creates a Matrix M. Since M doesn't cover peers' rankings with which each peer has never transacted, the ML-TFT introduces M^2 as: $(M^2)_{i,j} = M_{i,j} + \sum_n M_{i,n}M_{n,j}$, which also indirectly queries j's ranking from other peers. To precisely query all peers' rankings, the ML-TFT has to calculate M^k s.t. k is great enough, though M^2 covers 60 percent peers. The time complexity to calculate M^2 is $O(N^3)$ and $O(N^{k+1})$ for M^k, which is more expensive than the GC's approximately $O(N^2)$ complexity.

In [14], Habib and Chuang refer to how to map local scores of peers to global rankings (percentile ranks). They prove that with the 95 percent probability, each peer can calculate its percentile rank by obtaining only $O(\sigma^2)$ samples, where σ is the standard deviation of the original population of scores. In a distributed system, the GC algorithm guarantees to calculate 100 percent correct GCs by each peer's receiving all other peers' GCs which it has transacted with (uploaded to or downloaded from) before.

Give-to-Get [19] is an algorithm which prevents free-riding and balances each node's upload/download amounts in P2P Video-on-Demand. It splits video data into small chunks, and forces peers to keep the following rule: peers have to upload the chunks received from other peers to get additional chunks from those peers. Indeed, this works well in video streaming, but it will not be suitable for general file exchanges because uploading and downloading must be executed concurrently.

To compare the GC with such a mechanism that balances each node's upload/download amounts, we employ a simple "Upload/Download Balance" (UDB) technique in our simulation (Section 8.3). In the UDB, a peer is allowed to download from any other peer only if its upload amount $\geq$ download amount, which we think also reflects a classical incentive model. However, the UDB turns out to produce much greater rejection rates between cooperative users than the GC. The details are discussed in Section 8.3.

3 Design Considerations

3.1 Fair Situations in P2P Networks

In P2P networks, the fairest and most ideal situation will be: in every node, the total amount of upload = the total amount of download, such as ones shown in Fig. 2. However, in reality, some nodes only download from other nodes (free-riders), and some nodes mainly work as suppliers (contributors).

In order to make such an unfair network fairer, intuitively we may adopt the following rules:

1. Give high contribution nodes priority for downloading from other nodes.
2. Restrain low contribution nodes from downloading. (In other words, 1 and 2 mean to encourage upload to high contribution nodes rather than to low contribution nodes.)

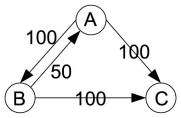

Fig. 3. Example of transactions: A uploaded 100 to B, 100 to C, and downloaded 50 from B, B uploaded 50 to A, 100 to C, and downloaded 100 from A, C uploaded 0, and downloaded 100 from B and 100 from A.

3. Encourage every node to download from low contribution nodes rather than from high contribution nodes.

3.2 Keys to Determining GCs

In consideration of the rules described above, we define the keys to determining the GC as follows:

1. A node which uploads much data and downloads little data, obtains a high contribution.
2. Upload to a high contribution node increases more contribution than upload to a low contribution node such as a free-rider.
3. Download from a high contribution (busy) node loses more contribution than download from a low contribution (free) node.

In the next section, we formally define the peer GC and the algorithm for calculating it on the basis of these keys.

4 Global Contribution

In this section, we first define the GC of a peer, then discuss the algorithms for computing it.

4.1 Proposed GC

On the basis of the keys described in Section 3.2, we propose to define the GC for each peer as:

$$x_i = \begin{cases} \alpha \frac{\beta \sum_{j\neq i} u_{ij}x_j + (1-\beta)\sum_{j\neq i} u_{ij} - \sum_{j\neq i} u_{ji}x_j}{\sum_{j\neq i}(u_{ij}+u_{ji})} + (1-\alpha), & \text{(i)} \\ \frac{2-\alpha(1+\beta)}{2+\alpha(1-\beta)}, & \text{(ii)} \end{cases}$$
$$\text{(i)} \sum_{j\neq i}(u_{ij}+u_{ji}) \neq 0, \qquad \text{(ii)} \sum_{j\neq i}(u_{ij}+u_{ji}) = 0, \tag{1}$$

where x_i is the GC of node i, u_{ij} (≥ 0) is the total amount of data uploaded from node i to j, $0 < \alpha < 1$ and $0 \leq \beta \leq 1$. Equation (1) can be converted to:

$$x_i = \begin{cases} \alpha \frac{\sum_{j\neq i}(\beta u_{ij} - u_{ji})x_j + (1-\beta)\sum_{j\neq i} u_{ij}}{\sum_{j\neq i}(u_{ij}+u_{ji})} + (1-\alpha), & \text{(i)} \\ \frac{2-\alpha(1+\beta)}{2+\alpha(1-\beta)}, & \text{(ii)} \end{cases} \tag{2}$$
$$\text{(i)} \sum_{j\neq i}(u_{ij}+u_{ji}) \neq 0, \qquad \text{(ii)} \sum_{j\neq i}(u_{ij}+u_{ji}) = 0.$$

We now provide the rationale for defining GC as in (1). For illustration, we use an example of a network consisting three nodes A, B, and C as shown in Fig. 3: A uploaded 100 to B, 100 to C, and downloaded 50 from B, B uploaded 50 to A, 100 to C, and downloaded 100 from A, C uploaded 0, and downloaded 100 from B and 100 from A.

First, without α and β, for $\sum_{j\neq i}(u_{ij}+u_{ji}) \neq 0$, the computation of GC becomes:

$$x_i = \frac{\sum_{j\neq i} u_{ij}x_j - \sum_{j\neq i} u_{ji}x_j}{\sum_{j\neq i}(u_{ij}+u_{ji})}. \tag{3}$$

If we apply (3) to node A's contribution, we have the following equation:

$$x_A = \frac{100x_B + 100x_C - 50x_B}{100+100+50}. \tag{4}$$

The denominator $\sum_{j\neq i}(u_{ij}+u_{ji})$ represents the summation of the total amounts of uploaded and downloaded data. Here, it's $100+100+50=250$. In the numerator, $\sum_{j\neq i} u_{ij}x_j$ is the summation of {(upload amount to node j) $\times$ (GC of j)}, which is the contribution gained by upload. Here, it is $100x_B + 100x_C$. This term captures the designed behavior stated in Section 3.2, namely, the GC of an uploading node will be higher if it uploads to a node with higher contribution than a node with lower contribution. $\sum_{j\neq i} u_{ji}x_j$ is the summation of {(download amount from node j) $\times$ (GC of j)}, which is the contribution subtracted by download. Here, it's $50x_B$. Since the negative of this term appears in the numerator, it reflects the designed behavior which encourages a node downloads from some nodes with low contributions. Roughly speaking, A's GC is determined by {contribution by upload} $-$ {contribution by download}.

Next, we use β $(0 \leq \beta \leq 1)$ as a free parameter to fine-tune the behavior of resource sharing. Intuitively, β represents a weight on how much the amount of upload to other nodes decides the overall upload contribution. In upload, we also have to consider the total upload amount, not only to which node the data have been transferred. Otherwise, a node which uploaded much data may earn a low contribution by sending only to low contribution nodes. (Indeed, upload to low contribution nodes should be avoided, but the fact that some amount of data was provided must be admitted.) Thus, to capture the generality, we have:

$$x_i = \frac{\beta \sum_{j\neq i} u_{ij}x_j + (1-\beta)\sum_{j\neq i} u_{ij} - \sum_{j\neq i} u_{ji}x_j}{\sum_{j\neq i}(u_{ij}+u_{ji})}, \tag{5}$$

and we have A's GC:

$$x_A = \frac{\beta(100x_B + 100x_C) + (1-\beta)(100+100) - 50x_B}{100+100+50}, \tag{6}$$

where the contribution by uploading to other nodes $(100x_B + 100x_C)$ is multiplied by β and a new contribution $(1-\beta) \times$ (the total upload amount), which is $(1-\beta)(100+100)$, is added to the contribution by upload.

α $(0 < \alpha < 1)$ mainly adjusts an initial (pregiven) GC value given to nodes which have no transaction histories. In practice, $\frac{2-\alpha(1+\beta)}{2+\alpha(1-\beta)}$ is the initial GC value and α partly determines it. The details of the initial GC value will be described later. Consequently, we have:

$$x_i = \alpha \frac{\beta \sum_{j\neq i} u_{ij}x_j + (1-\beta)\sum_{j\neq i} u_{ij} - \sum_{j\neq i} u_{ji}x_j}{\sum_{j\neq i}(u_{ij}+u_{ji})} + (1-\alpha) \tag{7}$$

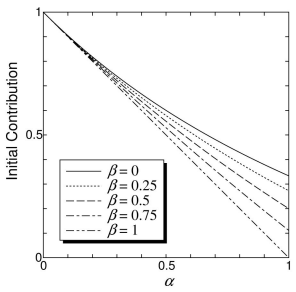

Fig. 4. Relationship among α, β, and initial GC value $\frac{2-\alpha(1+\beta)}{2+\alpha(1-\beta)}$: X is α and Y is $\frac{2-\alpha(1+\beta)}{2+\alpha(1-\beta)}$. α mainly determines $\frac{2-\alpha(1+\beta)}{2+\alpha(1-\beta)}$.

and

$$x_A = \alpha\frac{\beta(100x_B + 100x_C) + (1-\beta)(100+100) - 50x_B}{100+100+50} + (1-\alpha). \tag{8}$$

Each node can be assigned to individually unique α and β values. Also, it is possible to give pretrusted nodes extra credit by replacing $(1-\alpha)$ with $(1-\alpha)c_i$, where c_i is a weight for adjusting a GC value. This will help differentiate pretrusted nodes and others (see a similar research in [15, Section 4.5] of EigenTrust). However, to simplify the algorithm, we use the same α and β, and no c_i for all nodes in this paper.

Next, $\frac{2-\alpha(1+\beta)}{2+\alpha(1-\beta)}$ in (1) is the initial GC value given to beginner nodes which have no transaction histories. It is equal to the GC values obtained when the network is in the fairest situation, i.e., $x_1 = x_2 = \cdots = x_N$. (See Section 5.1 for details.) Notice that this value is used for keeping consistency with the fairest situations (Section 5.1) and with a dummy node (Section 7.1). Fig. 4 shows the relationship among α, β, and the initial GC value $\frac{2-\alpha(1+\beta)}{2+\alpha(1-\beta)}$.

Up until now, the GC of a node is defined in terms of the GCs of other nodes. Thus, to find a GC of a node, one can recursively find the GCs of all other nodes. In principle, this can be done by constructing a system of linear equations of the form in (1), each equation corresponds to computing GC of each node. Thus, all the GCs can be obtained by solving this system of linear equations. In particular, (9) is the matrix expression of (1) (assuming there is no beginner node).

$$Ax = b,$$

$$A = \begin{pmatrix} 1 & \frac{-\alpha(\beta u_{1,2}-u_{2,1})}{\sum_{j\neq 1}(u_{1,j}+u_{j,1})} & \cdots\cdots & \frac{-\alpha(\beta u_{1,N}-u_{N,1})}{\sum_{j\neq 1}(u_{1,j}+u_{j,1})} \\ \frac{-\alpha(\beta u_{2,1}-u_{1,2})}{\sum_{j\neq 2}(u_{2,j}+u_{j,2})} & 1 & \cdots\cdots & \frac{-\alpha(\beta u_{2,N}-u_{N,2})}{\sum_{j\neq 2}(u_{2,j}+u_{j,2})} \\ \vdots & \vdots & \vdots & \vdots \\ \frac{-\alpha(\beta u_{N,1}-u_{1,N})}{\sum_{j\neq N}(u_{N,j}+u_{j,N})} & \frac{-\alpha(\beta u_{N,2}-u_{2,N})}{\sum_{j\neq N}(u_{N,j}+u_{j,N})} & \cdots\cdots & 1 \end{pmatrix},$$

$$x = \begin{pmatrix} x_1 \\ \vdots \\ x_N \end{pmatrix}, \quad b = \begin{pmatrix} (1-\alpha) + \alpha(1-\beta)\frac{\sum_{j\neq 1} u_{1,j}}{\sum_{j\neq 1}(u_{1,j}+u_{j,1})} \\ \vdots \\ (1-\alpha) + \alpha(1-\beta)\frac{\sum_{j\neq N} u_{N,j}}{\sum_{j\neq N}(u_{N,j}+u_{j,N})} \end{pmatrix}. \tag{9}$$

Notice that A is a strictly diagonally dominant matrix (i.e., $|a_{ii}| > \sum_{j\neq i}|a_{ij}|, \forall i$. In (9), A's $|a_{ii}| = 1, \forall i$.), because

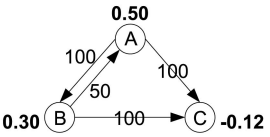

Fig. 5. Calculated GCs of model in Fig. 3.

$0 < \alpha < 1$. Therefore, (9) is guaranteed to converge with iterative methods such as the Jacobi, Gauss-Seidel, asynchronous iterative methods [2], [3], [4], [7] (see [24] Section 2 for the proof).

As a result, for $\alpha = 0.8,\ \beta = 0.5$, the GCs of the model in Fig. 3 can be expressed as (10) and we have the results shown in Fig. 5.

$$\begin{cases} x_A = 0.8 \times \frac{0.5 \times (100x_B + 100x_C) + 0.5 \times 200 - 50x_B}{200+50} + 0.2, \\ x_B = 0.8 \times \frac{0.5 \times (50x_A + 100x_C) + 0.5 \times 150 - 100x_A}{150+100} + 0.2, \\ x_C = 0.8 \times \frac{-(100A + 100x_B)}{0+200} + 0.2, \end{cases}$$
$$\Rightarrow \begin{cases} x_A = 0.501, \\ x_B = 0.301, \\ x_C = -0.120. \end{cases} \tag{10}$$

4.2 Algorithms for Computing GC

If a network is centralized, one can calculate all the GCs using the central machine. In particular, one can use some representative sequential iterative methods, such as the Jacobi, Gauss-Seidel methods. Algorithm 1 shows an example pseudocode for calculating all nodes' GCs with the Gauss-Seidel method. The calculation (iteration) should be stopped when all xs are no longer updated. Note that the time complexity of this algorithm is $O(N^2 \times (\#\text{ of iterations to converge}))$. Since the typical # of iterations to converge is turned out to be 10 from our simulation regardless of N (see Section 8.2), we could say the time complexity is approximately $O(N^2)$.

Algorithm 1. Pseudocode for calculating all GCs with Gauss-Seidel method

```
// Initialize all xs before calculating
for j = 1 to N do
    x_j = 0
end for

// Calculate all GCs
loop
    for i = 1 to N do
        tmp = u_up = u_down = 0.0
        for j = 1 to N do
            if j ≠ i and (u_ij ≠ 0 or u_ji ≠ 0) then
                tmp += (β u_ij − u_ji) x_j
                u_up += u_ij
                u_down += u_ji
            end if
        end for
        if u_up + u_down = 0 then
```

```
      x_i = (2 − α(1 + β))/(2 + α(1 − β))
    else
      x_i = α(tmp + (1 − β)u_up)/(u_up + u_down) + (1 − α)
    end if
  end for
end loop
```

In a decentralized network, distributed methods are more preferable. For example, the asynchronous iterative method [3] will be most suitable. In the asynchronous iterative method, each node calculates (1), then passes a new GC value to other nodes with which it has ever transacted. Algorithm 2 shows an example pseudocode with the asynchronous iterative method.

Algorithm 2. Pseudocode for calculating my GC x_i with asynchronous iterative method

```
// Initialize all xs before calculating
for j = 1 to N do
  x_j = 0
end for

// Calculate my GC x_i
loop
  tmp = u_up = u_down = 0.0
  for j = 1 to N do
    if j ≠ i and (u_ij ≠ 0 or u_ji ≠ 0) then
      tmp += (βu_ij − u_ji)x_j
      u_up += u_ij
      u_down += u_ji
    end if
  end for
  if u_up + u_down = 0 then
    x_i = (2 − α(1 + β))/(2 + α(1 − β))
  else
    x_i = α(tmp + (1 − β)u_up)/(u_up + u_down) + (1 − α)
  end if

  // Pass new x_i to other nodes
  for j = 1 to N do
    if j ≠ i and (u_ij ≠ 0 or u_ji ≠ 0) then
      Pass x_i to node j.
    end if
  end for

  Sleep a little to receive enough xs from other nodes
end loop
```

The pseudocode does not include receiving GCs from other nodes and how to stop the calculation (convergence check). Usually, it is better to receive GCs in a different thread. The calculation (iteration) should be stopped when GC x_i is no longer updated. However, in a real network, it will not happen very often because xs are updated as long as transactions occur. The time complexity in each node is approximately $O(N)$, because the # of iterations to converge is typically 10 as well (see Section 8.2).

As described in Section 4.1, the calculation is guaranteed to converge in both sequential and distributed methods, and it usually converges after a small number of iterations (see Section 8.2).

4.3 Estimate of New GCs

Before a transaction, peers will need to know how much GC they will gain/lose after the transaction, because they may prefer transacting with a node with which the loss of GC is minimum, or the gain of GC is maximum. We can approximate the new GC values after a transaction as follows: suppose q wants to download u' from p, then the approximate new GCs x'_p and x'_q are:

$$\begin{cases} x'_p = \\ \quad \alpha \dfrac{\beta\left\{\sum_{j\neq p,q} u_{pj}x_j + (u_{pq}+u')x'_q\right\} + (1-\beta)\left(\sum_{j\neq p} u_{pj} + u'\right) - \left(\sum_{j\neq p,q} u_{jp}x_j + u_{qp}x'_q\right)}{\sum_{j\neq p}(u_{pj}+u_{jp}) + u'} \\ \quad + (1-\alpha), \\ x'_q = \\ \quad \alpha \dfrac{\beta\left\{\sum_{j\neq q,p} u_{qj}x_j + u_{qp}x'_p\right\} + (1-\beta)\sum_{j\neq q} u_{qj} - \left\{\left(\sum_{j\neq q,p} u_{jq}x_j + (u_{pq}+u')x'_p\right)\right\}}{\sum_{j\neq q}(u_{qj}+u_{jq}) + u'} \\ \quad + (1-\alpha). \end{cases} \tag{11}$$

Equation (11) can be converted to:

$$\begin{cases} x'_p = \dfrac{\alpha\{\beta(u_{pq}+u') - u_{qp}\}}{\sum_{j\neq p}(u_{pj}+u_{jp}) + u'} x'_q + \\ \qquad \dfrac{\alpha\left\{\sum_{j\neq p,q}(\beta u_{pj} - u_{jp})x_j + (1-\beta)\left(\sum_{j\neq p} u_{pj} + u'\right)\right\}}{\sum_{j\neq p}(u_{pj}+u_{jp}) + u'} + (1-\alpha), \\ x'_q = \dfrac{\alpha\{\beta u_{qp} - (u_{pq}+u')\}}{\sum_{j\neq q}(u_{qj}+u_{jq}) + u'} x'_p + \\ \qquad \dfrac{\alpha\left\{\sum_{j\neq q,p}(\beta u_{qj} - u_{jq})x_j + (1-\beta)\sum_{j\neq q} u_{qj}\right\}}{\sum_{j\neq q}(u_{qj}+u_{jq}) + u'} + (1-\alpha). \end{cases} \tag{12}$$

Note that the calculation is executed only for two variables x'_p and x'_q. We use fixed values for other x_j, which are currently available on each node. Suppose:

$$\begin{cases} a_p = \dfrac{\alpha\{\beta(u_{pq}+u') - u_{qp}\}}{\sum_{j\neq p}(u_{pj}+u_{jp}) + u'}, \\ b_p = \dfrac{\alpha\left\{\sum_{j\neq p,q}(\beta u_{pj} - u_{jp})x_j + (1-\beta)\left(\sum_{j\neq p} u_{pj} + u'\right)\right\}}{\sum_{j\neq p}(u_{pj}+u_{jp}) + u'} + (1-\alpha), \\ a_q = \dfrac{\alpha\{\beta u_{qp} - (u_{pq}+u')\}}{\sum_{j\neq q}(u_{qj}+u_{jq}) + u'}, \\ b_q = \dfrac{\alpha\left\{\sum_{j\neq q,p}(\beta u_{qj} - u_{jq})x_j + (1-\beta)\sum_{j\neq q} u_{qj}\right\}}{\sum_{j\neq q}(u_{qj}+u_{jq}) + u'} + (1-\alpha), \end{cases} \tag{13}$$

then (12) can be expressed as:

$$\begin{cases} x'_p = a_p x'_q + b_p, \\ x'_q = a_q x'_p + b_q. \end{cases} \tag{14}$$

Consequently, we have:

$$\begin{cases} x'_p = \dfrac{a_p b_q + b_p}{1 - a_p a_q}, \\ x'_q = \dfrac{a_q b_p + b_q}{1 - a_p a_q}. \end{cases} \tag{15}$$

Each of p, q calculates both x'_p and x'_q. To do that, in a decentralized system, p must receive a_q and b_q from q, q must receive a_p and b_p from p. Thus, the approximate GC values are estimated.

In reality, update of x'_p and x'_q affects the entire network and will change other x_j. Therefore, we have to beware that x'_p and x'_q are not real values and will be slightly different from them. In our simulation, the estimated values are mostly close enough to the real ones.

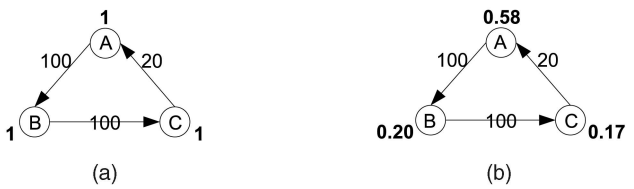

Fig. 6. Examples of contribution values calculated by (a) PageRank and EigenTrust, (b) GC for $\alpha = 0.8$, $\beta = 0.5$. Contributions of A, B, and C are supposed to be $x_A > x_B > x_C$. However, $x_A = x_B = x_C = 1$ with the PageRank and EigenTrust.

4.4 Comparison with PageRank and EigenTrust

In this section, we explain why the original PageRank and EigenTrust are not suitable for calculating global contributions in P2P networks.

The PageRank [25] ranks "popularity" of each web page using web links. The calculation of the PageRank is based on the following idea: if a web page is linked from another popular web page that has few links, then the linked web page becomes also popular. As well as the GC, the PageRank constructs a systems of linear equations and obtains each web page's ranking by solving it. (Due to the page limit, we do not describe the details of its algorithm.) However, as described in Section 3, the global contribution is based on a different idea of peer relationship, and therefore it is basically impossible to apply the principle of the PageRank to computing the global contribution. If we still adapt the PageRank [25] to calculating each peer's contribution using the transaction history, the equation will be:

$$x_i = \alpha \sum_{j \neq i} \frac{u_{ij}}{\sum_{k \neq j} u_{kj}} x_j + (1 - \alpha). \tag{16}$$

Applying it to a network model in Fig. 6, we have:

$$\begin{cases} x_A = \alpha \dfrac{100}{100} x_B + (1 - \alpha), \\ x_B = \alpha \dfrac{100}{100} x_C + (1 - \alpha), \\ x_C = \alpha \dfrac{20}{20} x_A + (1 - \alpha), \end{cases} \Rightarrow \begin{cases} x_A = 1, \\ x_B = 1, \\ x_C = 1. \end{cases} \tag{17}$$

However, intuitively the contributions must be

$$x_A > x_B > x_C,$$

because A has uploaded most and downloaded least, C has uploaded least and downloaded most. Thus, the PageRank is not suitable for computing the global contribution. On the other hand, the GC outputs reasonable values as seen in Fig. 6.

The EigenTrust is probably the closest algorithm to the GC. The EigenTrust also constructs a system of linear equations according to the relationship of peers based on "If a peer is relied on by another highly reliable peer, its reputation also becomes high." However, it does not count *minus* reliability. Suppose that a peer i rates transactions with a peer j as "10 satisfactory and 10 unsatisfactory transactions." Then, the i's local trust value toward j becomes $s_{ij} = 10 - 10 = 0$. Suppose that i rates transactions with another peer k as "20 unsatisfactory transactions." Then, the i's local trust value toward k becomes $s_{ik} = -20$. However, when a local trust value s is normalized as $max(s, 0)$, we have $max(s_{ij}, 0) = max(s_{ik}, 0) = 0$. As a result, even if we apply the EigenTrust to calculating the global contributions, it is hard to express the *minus* contribution lost by download. Should we adapt it to calculating each peer's contribution, then for example, we have the following equation:

$$x_i = \alpha \frac{\sum_{j \neq i} max(u_{ij} - u_{ji}, 0) x_j}{\sum_{j \neq i} max(u_{ij} - u_{ji}, 0)} + (1 - \alpha). \tag{18}$$

Applying it to a network model in Fig. 6, we have:

$$\begin{cases} x_A = \alpha \dfrac{max(100 - 0, 0) x_B + max(0 - 20, 0) x_C}{max(100 - 0, 0) + max(0 - 20, 0)} + (1 - \alpha), \\ x_B = \alpha \dfrac{max(0 - 100, 0) x_A + max(100 - 0, 0) x_C}{max(0 - 100, 0) + max(100 - 0, 0)} + (1 - \alpha), \\ x_C = \alpha \dfrac{max(20 - 0, 0) x_A + max(0 - 100, 0) x_B}{max(20 - 0, 0) + max(0 - 100, 0)} + (1 - \alpha), \end{cases}$$
$$\Rightarrow \begin{cases} x_A = 1, \\ x_B = 1, \\ x_C = 1. \end{cases} \tag{19}$$

Thus, we cannot use the EigenTrust as it is for calculating global contributions.

5 Theoretical Justification of GC

If a network is in the fairest situation (see Section 3.1 and Fig. 2), every node's GC value calculated based on the GC algorithm must be the same. Also, the GC algorithm is supposed to fulfill the keys to determining global contributions described in Section 3.2. In this section, we provide theoretical justification of the GC algorithm for the designed behaviors.

5.1 Fairest Situations

In Section 3.1, we described the fairest situation in P2P networks as: in every node, the total amount of upload = the total amount of download (see Fig. 2). In this case, every node's GC value must be the same. We prove that every node's GC value is the same *if and only if* the total amount of upload is equal to the total amount of download in every node.

- "The total amount of upload = the total amount of download in every node" means

$$\sum_{j \neq i} u_{ij} = \sum_{j \neq i} u_{ji}, \forall i.$$

- "Every node's contribution value is the same" means $x_1 = x_2 = \cdots = x_N$.

Before proving $\sum_{j \neq i} u_{ij} = \sum_{j \neq i} u_{ji}, \forall i \Leftrightarrow x_1 = \cdots = x_N$, we prove

$$x_1 = \cdots = x_N = x \Rightarrow x = \frac{2 - \alpha(1 + \beta)}{2 + \alpha(1 - \beta)}.$$

Proof of $x_1 = x_2 = \cdots = x_N = x \Rightarrow x = \frac{2-\alpha(1+\beta)}{2+\alpha(1-\beta)}$.

Suppose $x_1 = x_2 = \ldots = x_N = x$, then (2) becomes:

$$x = \begin{cases} \alpha \dfrac{\sum_{j\neq i}(\beta u_{ij} - u_{ji})x + (1-\beta)\sum_{j\neq i} u_{ij}}{\sum_{j\neq i}(u_{ij}+u_{ji})} + (1-\alpha), & \text{(i)} \\ \dfrac{2-\alpha(1+\beta)}{2+\alpha(1-\beta)}, & \text{(ii)} \end{cases} \quad (20)$$

$$\text{(i) } \sum_{j\neq i}(u_{ij}+u_{ji}) \neq 0, \qquad \text{(ii) } \sum_{j\neq i}(u_{ij}+u_{ji}) = 0.$$

Suppose $\sum_{j\neq i} u_{ij} = U_i$, $\sum_{j\neq i} u_{ji} = D_i$, then (20) can be converted to:

$$(U_i + D_i)(x - 1 + \alpha) = \alpha((\beta U_i - D_i)x + (1-\beta)U_i), \quad (21)$$

and as a result,

$$((1-\alpha\beta)U_i + (1+\alpha)D_i)x = (1-\alpha\beta)U_i + (1-\alpha)D_i. \quad (22)$$

Since $\sum_i U_i = \sum_i D_i\ (= T)$, we have:

$$\begin{aligned} \sum_i ((1-\alpha\beta)U_i + (1+\alpha)D_i)x &= \sum_i ((1-\alpha\beta)U_i + (1-\alpha)D_i) \\ ((1-\alpha\beta)T + (1+\alpha)T)x &= (1-\alpha\beta)T + (1-\alpha)T \\ T\left(x - \frac{2-\alpha(1+\beta)}{2+\alpha(1-\beta)}\right) &= 0. \end{aligned} \quad (23)$$

If $T = 0$, then $U_i = D_i = 0 \quad \forall i$, because $U_i, D_i \geq 0$. In this case, $x = \frac{2-\alpha(1+\beta)}{2+\alpha(1-\beta)}$ from (20). If $T \neq 0$, we have also $x = \frac{2-\alpha(1+\beta)}{2+\alpha(1-\beta)}$ because $x - \frac{2-\alpha(1+\beta)}{2+\alpha(1-\beta)} = 0$. Hence,

$$x_1 = x_2 = \cdots = x_N = x \Rightarrow x = \frac{2-\alpha(1+\beta)}{2+\alpha(1-\beta)}.$$

□

Proof of $\sum_{j\neq i} u_{ij} = \sum_{j\neq i} u_{ji}, \forall i \Rightarrow x_1 = x_2 = \cdots = x_N$**.**
Suppose $\sum_{j\neq i} u_{ij} = \sum_{j\neq i} u_{ji} = U_i$, then (2) becomes:

$$x_i = \begin{cases} \alpha \dfrac{\sum_{j\neq i}(\beta u_{ij} - u_{ji})x_j + (1-\beta)U_i}{2U_i} + (1-\alpha), & U_i \neq 0, \\ \dfrac{2-\alpha(1+\beta)}{2+\alpha(1-\beta)}, & U_i = 0. \end{cases} \quad (24)$$

If we substitute $x_1 = \cdots = x_N = \frac{2-\alpha(1+\beta)}{2+\alpha(1-\beta)}$ in (24), then

$$\begin{cases} x_i & = \dfrac{2-\alpha(1+\beta)}{2+\alpha(1-\beta)}, \\ \alpha \dfrac{\sum_{j\neq i}(\beta u_{ij} - u_{ji})x_j + (1-\beta)U_i}{2U_i} + (1-\alpha) & = \dfrac{2-\alpha(1+\beta)}{2+\alpha(1-\beta)}. \end{cases} \quad (25)$$

As a result, $x_1 = \cdots = x_N = \frac{2-\alpha(1+\beta)}{2+\alpha(1-\beta)}$ becomes a solution of (24). Since (24) has only one unique solution, $x_1 = \cdots = x_N = \frac{2-\alpha(1+\beta)}{2+\alpha(1-\beta)}$ becomes the unique solution. Hence,

$$\sum_{j\neq i} u_{ij} = \sum_{j\neq i} u_{ji}, \forall i \Rightarrow x_1 = x_2 = \cdots = x_N.$$

□

Proof of $x_1 = x_2 = \cdots = x_N \Rightarrow \sum_{j\neq i} u_{ij} = \sum_{j\neq i} u_{ji}, \forall i$**.**
If we substitute $x_1 = x_2 = \cdots = x_N = x = \frac{2-\alpha(1+\beta)}{2+\alpha(1-\beta)}$ in (22), we have:

$$\alpha(1-\alpha\beta)(U_i - D_i) = 0. \quad (26)$$

Since $\alpha \neq 0$ and $\alpha\beta \neq 1$, $U_i = D_i$ ($\sum_{j\neq i} u_{ij} = \sum_{j\neq i} u_{ji}$). Hence, $x_1 = x_2 = \cdots = x_N \Rightarrow \sum_{j\neq i} u_{ij} = \sum_{j\neq i} u_{ji}, \forall i$. □

5.2 Keys to Determining GCs

Before verifying if (1) fulfills the keys described in Section 3.2, we have to beware that a change of a node's GC influences a whole network and consequently changes all other nodes' GCs. As a result, it recursively affects the original node's GC. Therefore, it is not easy to strictly verify if (1) always follows the given conditions. Hence, we here discuss the approximate verification of (1).

1. Basically, a node, which uploads much data and downloads little data, obtains a high contribution:
   In (1), $\beta\sum_{j\neq i} u_{ij}x_j + (1-\beta)\sum_{j\neq i} u_{ij} - \sum_{j\neq i} u_{ji}x_j$ consists of
   $$\begin{aligned} &\{\beta \times (\text{upload amount to node } j) \times (\text{GC of } j) + (1-\beta) \\ &\quad \times (\text{total upload amount})\} \\ &\quad - \{(\text{download amount from node } j) \times (\text{GC of } j)\}. \end{aligned}$$
   Therefore, the greater the upload amounts are and the less the download amounts are, basically the greater $\beta\sum_{j\neq i} u_{ij}x_j + (1-\beta)\sum_{j\neq i} u_{ij} - \sum_{j\neq i} u_{ji}x_j$ becomes. Hence, we are able to say that, roughly, the more a node uploads and the less it downloads, the higher contribution is obtained.
2. Upload to a high contribution node increases more contribution than upload to a low contribution node such as a free-rider:
   In $\beta\sum_{j\neq i} u_{ij}x_j + (1-\beta)\sum_{j\neq i} u_{ij} - \sum_{j\neq i} u_{ji}x_j$, $\sum_{j\neq i} u_{ij}x_j$ represents the contribution obtained by upload to other nodes. For a certain k such that $u_{ik} \neq 0$, the greater x_k is, the greater $u_{ik}x_k$ becomes, in consequence, the greater $\sum_{j\neq i} u_{ij}x_j$ becomes and more contribution is gained.
3. Download from a high contribution (busy) node loses more contribution than download from a low contribution (free) node:
   In $\beta\sum_{j\neq i} u_{ij}x_j + (1-\beta)\sum_{j\neq i} u_{ij} - \sum_{j\neq i} u_{ji}x_j$, $\sum_{j\neq i} u_{ji}x_j$ represents the contribution subtracted by download. For a certain k such that $u_{ki} \neq 0$, the greater x_k is, the greater $u_{ki}x_k$ becomes, in consequence, the greater $\sum_{j\neq i} u_{ji}x_j$ becomes and more contribution is lost.

Thus, (1) approximately follows the keys in Section 3.2.

6 Transaction and Keeping Fairness

In this section, we discuss transaction procedures using the GC and explain how they keep a network fair.

Among several possible transaction procedures, we introduce the following two procedures:

Transaction Procedure 1

1. Search for uploaders which have files a downloader wants and whose GC values $\leq$ the downloader's. (In other words, the downloader cannot download from those whose GC values $>$ the downloader's.)
2. For each of the chosen uploaders, estimate the GC values which will be gained after the transaction using (11).
3. For every uploader, if its estimated GC value after the transaction is lower than the current one, then reject the transaction. (All the uploaders will lose their contributions by uploading to the downloaders.)

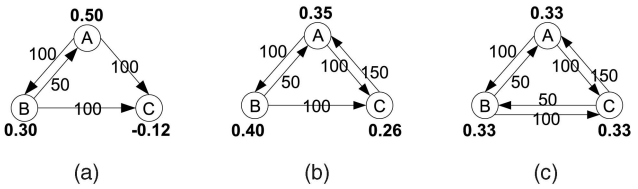

Fig. 7. Example of transactions using GC: C wants to download 100, A wants to download 150, then B wants to download 50 from somebody. (a) Before transaction. (b) After A has downloaded 150 from C. (c) After B has downloaded 50 from C.

4. Choose an uploader with which the estimated loss of the downloader's GC value is minimum.

Transaction Procedure 2

1. If the downloader's GC value is lower than a certain value (rejection threshold), for example 0, then the transaction is rejected unconditionally. Otherwise, go to step 2.
2. Search for uploaders which have files the downloader wants.
3. Do the same as 2 in transaction procedure 1.
4. Do the same as 3 in transaction procedure 1.
5. Do the same as 4 in transaction procedure 1.

For example, in Fig. 7a, suppose that C wants to download 100, A wants 150 and B wants 50 in this order, and that every node always has a file other nodes want. With transaction procedure 1, first C is rejected because its GC value is less than all other nodes'. Next, A estimates its GC value with B and C. With B, A's estimated GC value becomes 0.24. With C, it becomes 0.34. Since A loses less GC with C, A chooses C to transact. After A has downloaded 150 from C, then we have the GCs as in Fig. 7b. Next, B estimates its GC value with A and C. With A, B has 0.30, and with C, B has 0.33. Consequently, B chooses C. After the transactions, we have the GCs as in Fig. 7c. No uploaders have lower estimated values.

With transaction procedure 2, suppose the rejection threshold is 0, C is rejected because its GC value $-0.12 < 0$. Next, A estimates its GC value with B and C, and chooses C as well as transaction procedure 1. Then, B also chooses C as well.

In these procedures, free-riding can be avoided by step 1 (provided that an appropriate threshold value is selected in transaction procedure 2). Since, with just one first download, a free-rider loses much contribution, further free-riding can be prevented. Mostly, this condition continues to hold until the free-rider uploads to somebody. Thus, we can efficiently prevent free-riding. Also, using steps 2 and 4 in transaction procedure 1 or 3 and 5 in transaction procedure 2, a network naturally tends to a fairer situation without forcing peers to cooperate with each other. Thus, we can easily maintain fairness in the network.

Besides, it is possible to let downloaders choose uploaders without estimating their GC values as long as the restriction that a downloader can download only from equal or lower contribution nodes or the rejection threshold is adopted. For example, downloaders may prefer uploaders with wide bandwidths. However, in this case, if a downloader downloads from a high contribution node, it will lose much GC, and therefore it will have to upload much to restore its GC.

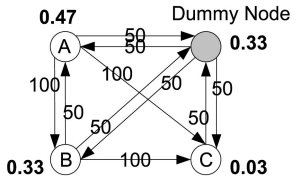

Fig. 8. Example of network model using dummy node.

Thus, there are various possible transaction procedures. Finding the best procedure is our important future work.

7 Considerations for Practical Usage

7.1 Adjusting Initial Condition

Since the GC is mainly decided by the ratio of amounts of upload and download, beginner nodes may lose much contribution by downloading only a small amount of data. This can be compared to batting averages in the beginning of a baseball season; they drastically change by a single strikeout or a hit. In practical use, some adjustment may be necessary. This can be achieved by introducing a dummy node which has a constant GC value and the same constant amounts of upload and download toward every node in a network. Fig. 8 shows an example network model with a dummy node based on the model in Fig. 3 (Notice $\alpha = 0.8$, $\beta = 0.5$). The dummy node always has a constant GC value $\frac{2-\alpha(1+\beta)}{2+\alpha(1-\beta)}$ which is equal to the initial GC value (see Section 4.1), and has constant dummy amounts of upload and download with every node. In Fig. 8, the dummy amounts of upload and download are 50 and the GC value of the dummy node is 0.33 ($= \frac{2-\alpha(1+\beta)}{2+\alpha(1-\beta)}$ for $\alpha = 0.8$, $\beta = 0.5$). Compared with Fig. 5, C's GC value is improved from -0.12 to 0.03.

Suppose the dummy amount is $\hat{u}$, then the algorithm for adjusting the initial condition becomes:

$$x_i = \alpha \Bigg\{ \beta \sum_{j \neq i} u_{ij} x_j + (1-\beta) \sum_{j \neq i} u_{ij} - \sum_{j \neq i} u_{ji} x_j + \beta \hat{u} \frac{2-\alpha(1+\beta)}{2+\alpha(1-\beta)} + (1-\beta)\hat{u} - \hat{u} \frac{2-\alpha(1+\beta)}{2+\alpha(1-\beta)} \Bigg\} \Bigg/ \Bigg\{ \sum_{j \neq i} (u_{ij} + u_{ji}) + 2\hat{u} \Bigg\} + (1-\alpha), \quad (27)$$

or

$$x_i = \alpha \frac{\sum_{j \neq i} (\beta u_{ij} - u_{ji}) x_j + (1-\beta) \sum_{j \neq i} u_{ij} + \frac{2\alpha(1-\beta)}{2+\alpha(1-\beta)} \hat{u}}{\sum_{j \neq i} (u_{ij} + u_{ji}) + 2\hat{u}} + (1-\alpha). \quad (28)$$

Note $x_i = \frac{2-\alpha(1+\beta)}{2+\alpha(1-\beta)}$ still holds for all i in the fairest situations (see Section 5.1).

7.2 Other Considerations

In this section, we propose solutions for two other considerations, which require further investigations.

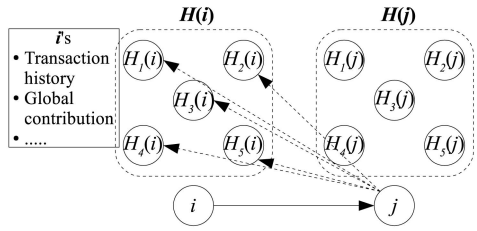

Fig. 9. Example of calculator nodes: j wants to download from i, then j inquires of i's calculator nodes ($H_1(i),\ H_2(i),\ \ldots.$) i's GC.

7.2.1 Security

While the paper does not investigate the security issues, they are very important. In this section, we focus mainly on false reports of GCs and study how to cope with it. In a decentralized system, each peer must report its own GC. Therefore, the false report of the GC arises as a security issue. To cope with it, we apply the following technique introduced by EigenTrust [15]:

- Some nodes may lie. Therefore, instead of each node's calculating its own GC, multiple other nodes (calculator nodes) calculate it. (See Fig. 9. $H_1(i)$, $H_2(i), \ldots$ are i's calculator nodes.)
- The calculator nodes can be chosen by a distributed hash table (DHT) algorithm, such as CAN [21] or Chord [23]. Multiple calculator nodes can be decided by different hash tables ($H_1,\ H_2,\ \ldots$).
- The calculator nodes must not know for which node they are calculating the GC, because they may cheat and report false GC values on purpose.
- The transaction history must be stored by the calculator nodes, instead of the host node itself.
- Suppose that node j wants to download from i and that j wants to check i's GC. Then, j inquires of i's calculator nodes ($H_1(i),\ H_2(i),\ \ldots$), and picks up the major value. (i's calculator nodes may return different GC values; therefore, a "majority vote" is adopted.)
- To calculate i's GC x_i, i's all calculator nodes $H_1(i),\ H_2(i),\ \ldots$ execute (1), then send x_i to $H_1(k),\ H_2(k),\ \ldots$, for all k such that either u_{ik} or $u_{ki} \neq 0$. Repeat it until x_i is not updated any more (see also Section 4.2).

Notice that this is not necessary in a centralized system because the central machine calculates and reports every node's GC.

Another important issue is "collusion," that is false reports of transactions. The GC algorithm is still robust against it, because in most transactions, an uploader gains GC and a downloader loses GC. It is hard for both the uploader and downloader to profit at the same time.

In addition, there remain more issues in our case, such as how to safely store each node's transaction history in a decentralized system. Those are under investigation and will be our future focus.

7.2.2 Other Factors to Decide Contribution

Equation (1) focuses only on the amount of upload/download. However, some files are very important or rare even if their sizes are small. Therefore, we may also need to take account of the number of files uploaded/downloaded.

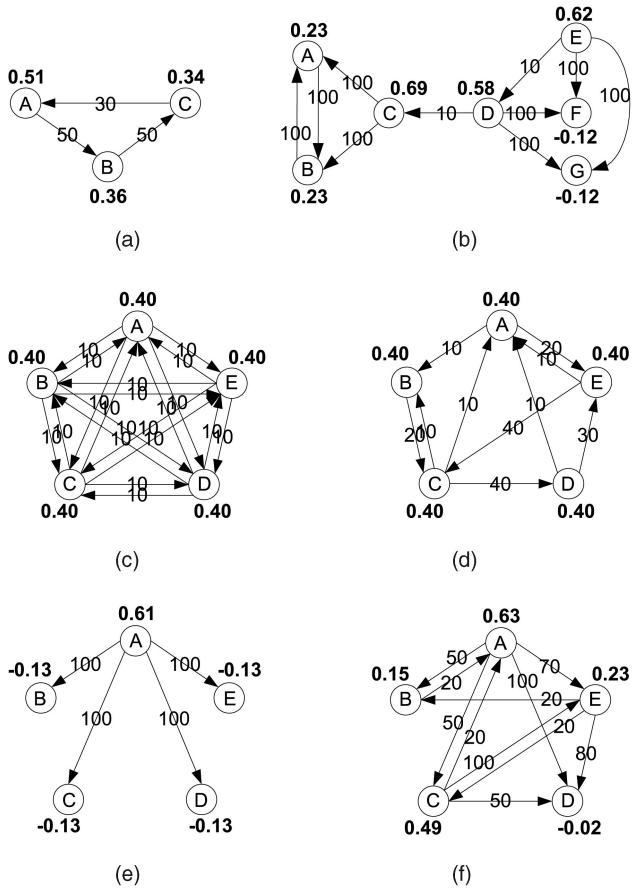

Fig. 10. Simulation results of P2P network models for $\alpha = 0.7,\ \beta = 0.5$.

In this case, we will be able to obtain the correspondent GC by replacing u_{ij} in (1) with $(u_{ij} + \delta f_{ij})$, where f_{ij} is the number of files uploaded from i to j, δ is a coefficient (or weight) for f_{ij}. The same technique will also be applicable to other factors which will affect the GC.

In this manner, our algorithm can pliantly cope with various different factors. Finding what factors affect the GC will be indispensable for improving our algorithm.

8 Simulation Results

In this section, we show simulation results for GC values in some network models, calculation convergence, and transactions. The simulation has been made with our original programs, which are specifically written for the GC and do not cover all the considerations needed to simulate real P2P networks, such as network topology, search algorithms.

8.1 Network Model

Fig. 10 shows simulation results of GC values in small networks, and Table 1 partially shows ones in a large network randomly generated for 10,000 nodes, ordered by contribution value. All models are calculated for $\alpha = 0.7,\ \beta = 0.5$. As a whole, the GC algorithm is shown to produce intuitively understandable contribution values.

8.2 Convergence

We simulated the calculation of (9) for randomly generated networks for 10,000 nodes, with a single computer and with eight computers. The single computer simulates a centralized

TABLE 1
Simulation Result of Contribution Values for 10,000 Nodes

Order	Contribution val.	Total upload amnt.	Total download amnt.
1	0.587	20163	6722
2	0.584	22775	7533
3	0.583	22579	7780
...			
5000	0.389	8806	8903
5001	0.389	11355	11564
...			
9998	-0.006	0	12346
9999	-0.007	0	9894
10000	-0.009	0	9600

system, and we used the Gauss-Seidel method to calculate the GCs (see Algorithm 1). The eight computers simulate a decentralized system, and we used the asynchronous iterative method (see Algorithm 2). Figs. 11a and 11b show example results of residuals in total GC values, respectively, with a single computer and with eight computers for $\alpha = 0.7,\ \beta = 0.5$. In our simulation, the calculation usually converges within 10 iterations in both systems, for any size of P2P network. (Notice that in the beginning stage of a network, i.e., if there have been few transactions made, the calculation may take more iterations, such as 30 or 40.)

8.3 Transaction

In Section 6, we described how efficiently the GC algorithm prevents free-riding and keeps fairness in a network. We now verify it via simulation results. The simulated transactions are as follows:

- The number of nodes is 1,000.
- Free-riders never share their files.
  - Out of 1,000 nodes, free-riders are randomly chosen according to the free-rider rate. The free-rider rates used in the simulation are 0, 30, 50, and 70 percent.
  - A downloader is randomly chosen out of the 1,000 nodes, and the download data amount is randomly chosen between 1 and 255.
  - To start a transaction, first, the downloader searches for non free-riders which have files the downloader wants. Here, we suppose each non free-rider owns the file with a probability of 10 percent. In other words, on average, 10 percent of non free-riders have the files the downloader wants.
  - Out of the non free-riders which have the files, we choose one uploader as follows:

    No restrictions. Choose the closest node in the node number.

    Tit-for-tat. Choose a node such that # of times the downloader has uploaded to the node $\geq$ # of times the node has uploaded to the downloader (simple tit-for-tat), and which is closest to the downloader in the node number.

    Upload/download balance (UDB). If the downloader's total uploaded amount $\geq$ its total downloaded amount, choose the closest node in the node number. Otherwise, reject the transaction.

    GC. Follow the two transaction procedures described in Section 6 (GC1, GC2, respectively). The rejection threshold in GC2 is 0.
  - Download data from the uploader. Afterwards, recalculate all nodes' GC values.
  - Repeat 2-5 1,000,000 times, that is, make 1,000,000 transactions.
  - Do the above procedure for four different free-rider rates: 0, 30, 50, and 70 percent, and five different techniques: no restrictions, tit-for-tat, UDB, the GC1 with $\alpha = 0.8$, 0.5, 0.2 and $\beta = 0.8$, 0.5, 0.2, and the GC2 with $\alpha = 0.9$, 0.8, 0.7 and $\beta = 0.2$, 0.5, 0.8.

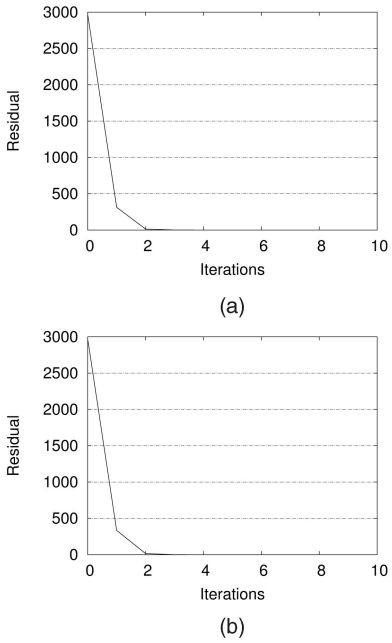

Fig. 11. Examples of calculation convergence: the calculation usually converges within 10 iterations both in a centralized and in a decentralized system. (a) Centralized system (with a single computer). (b) Decentralized system (with eight computers).

In our simulation, in order to compare the GC with classical incentive mechanisms (or the Give-to-Get [19], see Section 2) that balance the upload/download amounts in each node, we include the UDB. Since in the UDB, a peer cannot download from anybody until its upload amount exceeds or equals its download amount, it balances the upload/download amounts in each node.

Fig. 12 shows the times (out of 1,000,000 transactions) the data have been downloaded: by non free-riders, by free-riders and rejected by all uploaders. As seen, the GCs, except for the GC2 with $\alpha = 0.7,\ \beta = 0.5$ and 0.8, efficiently block the data transfer to free-riders, with no noticeable effect on the transactions between non free-riders. In those GCs, the numbers of downloads by free-riders are a little larger than the populations of free-riders, although they hardly appear on the graphs. Once a free-rider downloads from a non free-rider, its GC value usually becomes very low. Since those free-riders' GC values hardly recover, they are mostly rejected in all subsequent transactions. Thus, the GC efficiently restrains free-riding.

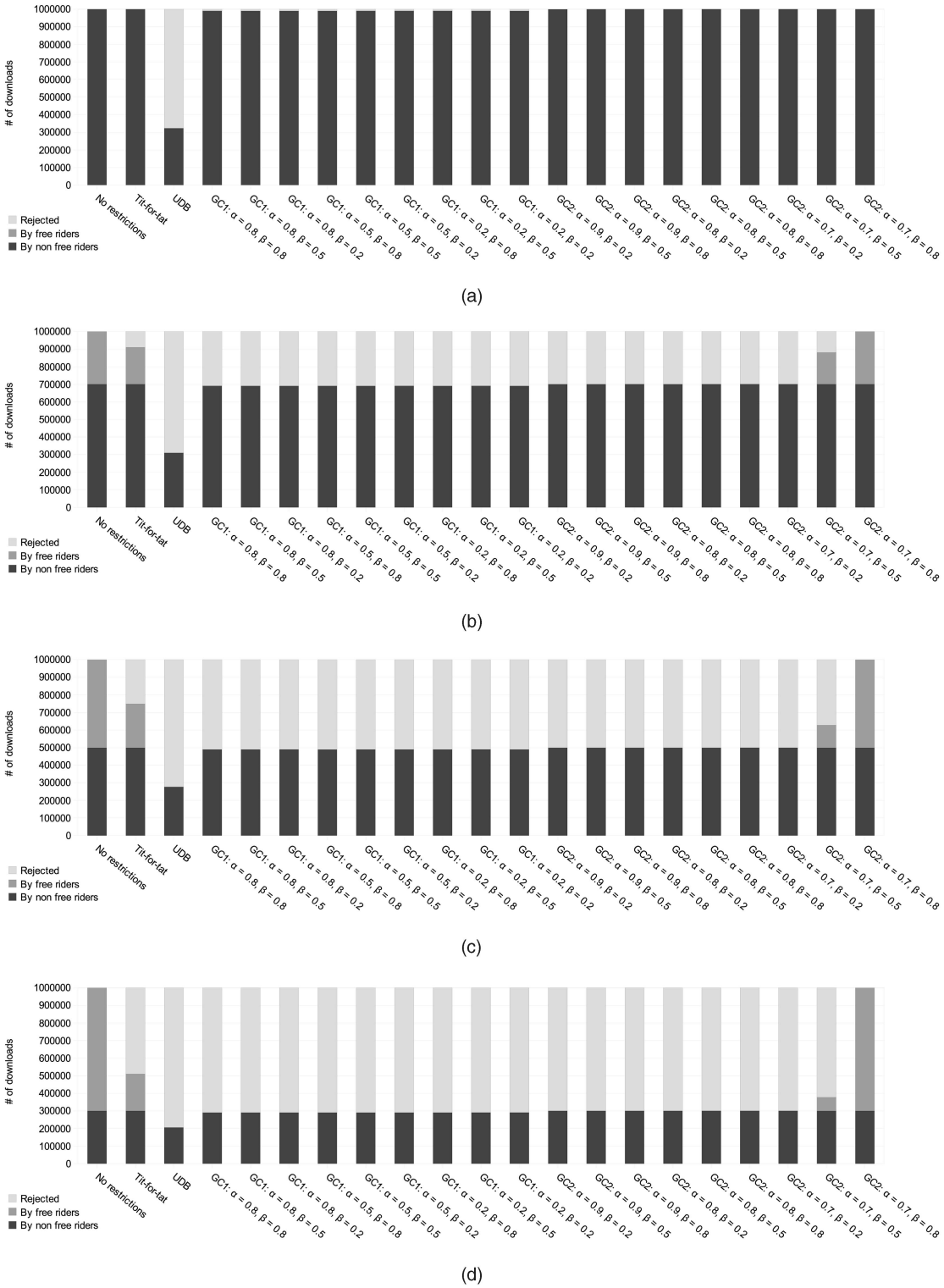

Fig. 12. # of downloads: by non free-riders, by free-riders and rejected (1,000 nodes and 1,000,000 total transactions). The GC with proper choice of α and β is effective in preventing free-riding without affecting transactions between non free-riders. Tit-for-tat does not prevent all free-riding. The UDB prevents free-riding but rejects some transactions between non free-riders. (a) Free-rider rate $= 0$ percent. (b) Free-rider rate $= 30$ percent. (c) Free-rider rate $= 50$ percent. (d) Free-rider rate $= 70$ percent.

The GC2 with $\alpha = 0.7$ with $\beta = 0.8$ does not work well. This is due to inappropriate choice of the rejection threshold ($= 0$). For $\alpha = 0.7$ and $\beta = 0.8$, the GC value seldom becomes lower than 0.

Tit-for-tat does not prevent free-riding as efficiently as the GCs. The UDB prevents free-riding as efficiently as the GCs. However, the rejection between non free-riders is conspicuous. With 0 percent free-riders (i.e., every peer is cooperative), about 67 percent of transactions are rejected, while the GCs reject only 1 percent (Fig. 12). With 50 percent free-riders, 45 percent of transactions between non free-riders are rejected, while the GCs reject 2 percent (Fig. 12). This

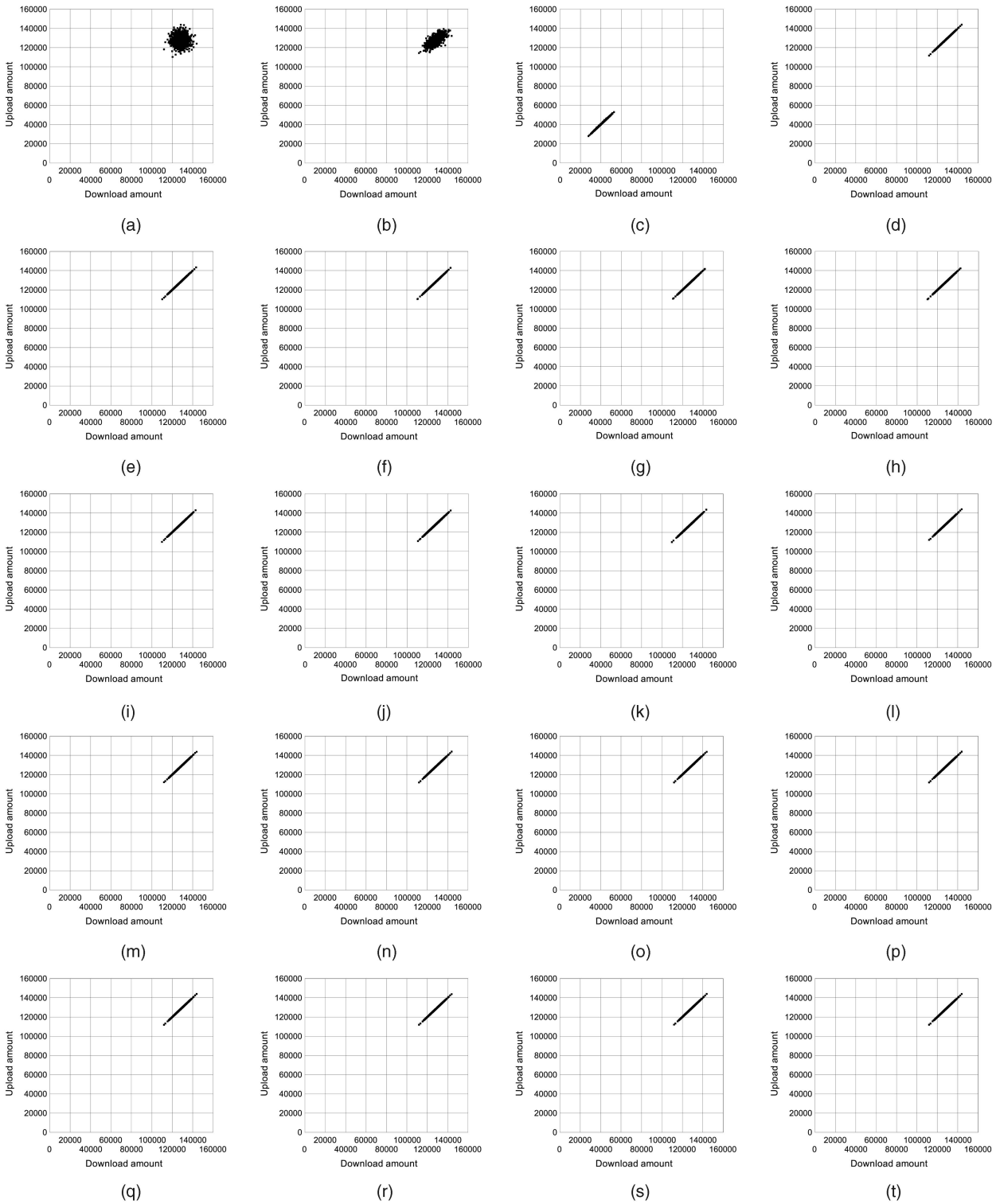

Fig. 13. Download and upload amounts in each node (free-rider rate = 0 percent). The GCs balance the download/upload amounts in each node. Tit-for-tat does not balance well. The UDB balances well but the transacted amounts are small. (a) No restrictions, (b) tit-for-tat, (c) UDB, (d) GC1: $\alpha = 0.8$, $\beta = 0.8$, (e) GC1: $\alpha = 0.8$, $\beta = 0.5$, (f) GC1: $\alpha = 0.8$, $\beta = 0.2$, (g) GC1: $\alpha = 0.5$, $\beta = 0.8$, (h) GC1: $\alpha = 0.5$, $\beta = 0.5$, (i) GC1: $\alpha = 0.5$, $\beta = 0.2$, (j) GC1: $\alpha = 0.2$, $\beta = 0.8$, (k) GC1: $\alpha = 0.2$, $\beta = 0.5$, (l) GC2: $\alpha = 0.9$, $\beta = 0.2$, (m) GC2: $\alpha = 0.9$, $\beta = 0.5$, (n) GC2: $\alpha = 0.9$, $\beta = 0.8$, (o) GC2: $\alpha = 0.8$, $\beta = 0.2$, (p) GC2: $\alpha = 0.8$, $\beta = 0.5$, (q) GC2: $\alpha = 0.8$, $\beta = 0.8$, (r) GC2: $\alpha = 0.7$, $\beta = 0.2$, (s) GC2: $\alpha = 0.7$, $\beta = 0.5$, (t) GC2: $\alpha = 0.7$, $\beta = 0.8$.

happens because in the UDB a downloader is not obligated to choose a specific uploader such that its upload amount is far less than the download amount. In the GC transaction, a downloader chooses an uploader with whom the estimated loss of the downloader's GC after the transaction is minimum. Therefore, the downloader most likely picks up an uploader whose GC is the lowest. As a result, the uploader instantly restores its GC and can be prepared for its next download.

So far, we have shown how efficiently the GC prevents free-riding. Next, we show how fair a network is kept with the GC. Figs. 13 and 14 plot the total download and upload amounts in each node, respectively, for the free-rider rate 0

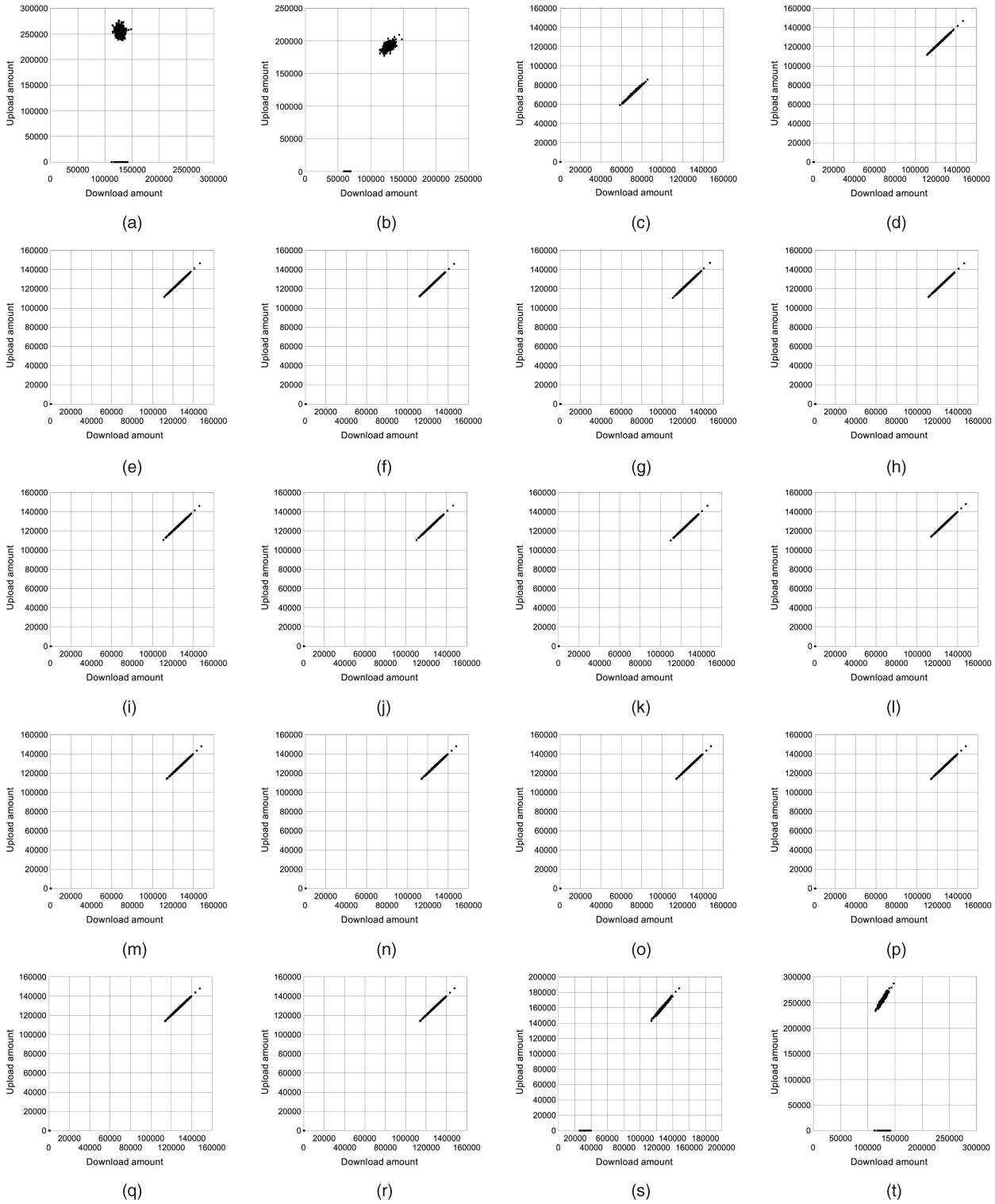

Fig. 14. Download and upload amounts in each node (free-rider rate = 50 percent). The GCs except for (s) and (t) balance the download/upload amounts in each node. Tit-for-tat does not balance well. The UDB balances well but the transacted amounts are small. (a) No restrictions, (b) tit-for-tat, (c) UDB, (d) GC1: $\alpha = 0.8$, $\beta = 0.8$, (e) GC1: $\alpha = 0.8$, $\beta = 0.5$, (f) GC1: $\alpha = 0.8$, $\beta = 0.2$, (g) GC1: $\alpha = 0.5$, $\beta = 0.8$, (h) GC1: $\alpha = 0.5$, $\beta = 0.5$, (i) GC1: $\alpha = 0.5$, $\beta = 0.2$, (j) GC1: $\alpha = 0.2$, $\beta = 0.8$, (k) GC1: $\alpha = 0.2$, $\beta = 0.5$, (l) GC2: $\alpha = 0.9$, $\beta = 0.2$, (m) GC2: $\alpha = 0.9$, $\beta = 0.5$, (n) GC2: $\alpha = 0.9$, $\beta = 0.8$, (o) GC2: $\alpha = 0.8$, $\beta = 0.2$, (p) GC2: $\alpha = 0.8$, $\beta = 0.5$, (q) GC2: $\alpha = 0.8$, $\beta = 0.8$, (r) GC2: $\alpha = 0.7$, $\beta = 0.2$, (s) GC2: $\alpha = 0.7$, $\beta = 0.5$, (t) GC2: $\alpha = 0.7$, $\beta = 0.8$.

and 50 percent. (Due to the space limit, we do not put all the results.) As described in Section 3.1, the fairest situation in P2P networks is: in every node, the total amount of upload equals to the total amount of download. As seen, in the GCs, except for the GC2 with $\alpha = 0.7,\ \beta = 0.5$ and 0.8, the download and upload amounts in non free-riders are well balanced. For the free-rider rate $= 0$ percent, they are almost equal in every node. Tit-for-tat does not balance as well as the GC, though it does better than no restrictions. The UDB also balances well although the transacted data amounts are less than those of the GCs.

TABLE 2
Fairness Metric Values (F in (30))

Free rider rate	0%	30%	50%	70%
No restrictions	0.02053	0.42344	0.66704	0.86154
Tit-for-tat	0.01116	0.39139	0.60025	0.77782
UDB	0.00134	0.30089	0.50073	0.70060
GC1: α=0.8, β=0.8	0.00023	0.30028	0.50032	0.70037
GC1: α=0.8, β=0.5	0.00024	0.30026	0.50031	0.70036
GC1: α=0.8, β=0.2	0.00022	0.30024	0.50032	0.70036
GC1: α=0.5, β=0.8	0.00023	0.30027	0.50031	0.70036
GC1: α=0.5, β=0.5	0.00024	0.30025	0.50031	0.70035
GC1: α=0.5, β=0.2	0.00023	0.30027	0.50031	0.70036
GC1: α=0.2, β=0.8	0.00033	0.30028	0.50032	0.70036
GC1: α=0.2, β=0.5	0.00031	0.30028	0.50032	0.70036
GC1: α=0.2, β=0.2	0.00029	0.50032	0.50032	0.70036
GC2: α=0.9, β=0.2	0.00023	0.30022	0.50027	0.70034
GC2: α=0.9, β=0.5	0.00023	0.30023	0.50028	0.70034
GC2: α=0.9, β=0.8	0.00024	0.30026	0.50031	0.70035
GC2: α=0.8, β=0.2	0.00023	0.30022	0.50028	0.70034
GC2: α=0.8, β=0.5	0.00023	0.30024	0.50029	0.70034
GC2: α=0.8, β=0.8	0.00024	0.30037	0.50044	0.70045
GC2: α=0.7, β=0.2	0.00022	0.30025	0.50032	0.70037
GC2: α=0.7, β=0.5	0.00025	0.38001	0.55716	0.73431

In order to measure fairness, we define the following metric:

$$F = \frac{1}{N}\sum_{i}^{N}\frac{\left|\sum_{j\neq i}^{N} u_{ij} - \sum_{j\neq i}^{N} u_{ji}\right|}{\sum_{j\neq i}^{N} u_{ij} + \sum_{j\neq i}^{N} u_{ji}}. \quad (29)$$

Since $\sum_{j\neq i}^{N} u_{ij} = U_i$ (the total upload amount in node i) and $\sum_{j\neq i}^{N} u_{ji} = D_i$ (the total download amount), (29) can be expressed as:

$$F = AVERAGE_i\left(\frac{|U_i - D_i|}{U_i + D_i}\right). \quad (30)$$

Notice that $F \geq 0$ and the smaller the metric value F is, the fairer the network is.

Table 2 shows the metric values in all techniques and all free-rider rates. As the results show, the metric values in the GC with appropriate choice of α and β are always better than those of tit-for-tat. This also backs up that the GC keeps a network fairer than tit-for-tat.

9 Conclusion and Future Work

In this paper, we introduced the GC algorithm which calculates each peer's global contribution in a P2P network, and explained how to use it for transactions. The algorithm is simple but powerful, and it efficiently keeps a network in a fair condition.

However, it still requires much work to improve security and to investigate other transaction procedures. Furthermore, we have not yet grasped all issues which will arise in practical usage. Therefore, experiments in real P2P networks will be indispensable in the future. Also, as described in Section 7.2.2, we will need to consider other factors to determine global contributions.

Acknowledgments

This work is supported in part by the US National Science Foundation (NSF) grants: CNS-0834775, CAREER-CNS-0845476, and by ASUSA Corporation, Salem, OR 97301.

References

[1] K.G. Anagnostakis, F. Harmantzis, S. Ioannidis, and M. Zghaibeh, "On the Impact of Practical P2P Incentive Mechanism on User Behavior," NET Institute Working Paper No. 06-14, Sept. 2006.
[2] J.M. Bahi, S. Contassot-Vivier, and R. Couurier, *Parallel Iterative Algorithms.* Chapman & Hall/CRC, 2007.
[3] G.M. Baudet, "Asynchronous Iterative Methods for Multiprocessors," *J. ACM,* vol. 25, no. 2, pp. 226-244, 1978.
[4] D.P. Bertsekas and J.N. Tsitsiklis, *Parallel and Distributed Computation: Numerical Methods.* Athena Scientific, 1997.
[5] A.R. Bharambe, C. Herley, and V.N. Padmanabhan, "Analyzing and Improving BitTorrent Performance," Technical Report MSR-TR-2005-03, Microsoft Research, Microsoft Corporation, 2005.
[6] C. Buragohain, D. Agrawal, and S. Suri, "A Game-Theoretic Framework for Incentives in P2P Systems," *Proc. Third Int'l Conf. Peer-to-Peer Computing,* pp. 48-56, Sept. 2003.
[7] D. Chazan and W. Miranker, "Chaotic Relaxation," *Linear Algebra and its Applications,* vol. 2, pp. 199-222, 1969.
[8] K. Eger and U. Killat, "Fair Resource Allocation in Peer-to-Peer Networks (Extended Version)," *Computer Comm.,* vol. 30, no. 16, pp. 3046-3054, Nov. 2007.
[9] M. Feldman and J. Chuang, "Overcoming Free-Riding Behavior in Peer-to-Peer Systems," *ACM SIGecom Exchanges,* vol. 5, no. 4, pp. 41-50, July 2005.
[10] M. Feldman, K. Lai, I. Stoica, and J. Chuang, "Robust Incentive Techniques for Peer-to-Peer Networks," *Proc. Fifth ACM Conf. Electronic Commerce,* pp. 102-111, May 2004.
[11] M. Feldman, C. Papadimitriou, J. Chuang, and I. Stoica, "Free-Riding and Whitewashing in Peer-to-Peer Systems," *Proc. ACM SIGCOMM Workshop Practice and Theory of Incentives in Networked Systems,* pp. 228-236, 2004.
[12] P. Garbacki and D.H.J. Epema, "An Amortized Tit-For-Tat Protocol for Exchanging Bandwidth Instead of Content in P2P Networks," *Proc. First Int'l Conf. Self-Adaptive and Self-Organizing Systems,* pp. 119-228, 2007.
[13] P. Golle, K. Leyton-Brown, and I. Mironov, "Incentives for Sharing in Peer-to-Peer Networks," *Proc. Third ACM Conf. Electronic Commerce,* pp. 75-78, 2001.
[14] A. Habib and J. Chuang, "Incentive Mechanism for Peer-to-Peer Media Streaming," *Proc. 12th IEEE Int'l Workshop Quality of Service (IWQOS '04),* pp. 171-180, 2004.
[15] S.D. Kamvar, M.T. Schlosser, and H. Garcia-Molina, "The EigenTrust Algorithm for Reputation Management in P2P Networks," *Proc. 12th Int'l World Wide Web Conf.,* 2003.
[16] K. Lai, M. Feldman, I. Stoica, and J. Chuang, "Incentives for Cooperation in Peer-to-Peer Networks," *Proc. Workshop Economics of Peer-to-Peer Systems,* June 2003.
[17] Q. Lian, Y. Peng, M. Yang, Z. Zhang, Y. Dai, and X. Li, "Robust Incentives via Multi-Level Tit-for-Tat," *Concurrency and Computation: Practice & Experience,* vol. 20, pp. 167-178, 2008.
[18] J.J.D. Mol, J.A. Pouwelse, D.H.J. Epema, and H.J. Sips, "Free-Riding, Fairness, and Firewalls in P2P File-Sharing," *Proc. Eighth Int'l Conf. Peer-to-Peer Computing,* pp. 301-310, Sept. 2008.
[19] J.J.D. Mol, J.A. Pouwelse, D.H.J. Epema, and H.J. Sips, "Give-to-Get: Free-Riding Resilient Video-on-Demand in P2P Systems," *Proc. Multimedia Computing and Networking,* pp. 681804-1-681804-8, 2008.
[20] T. Ngan, D.S. Wallach, and P. Druschel, "Enforcing Fair Sharing of Peer-to-Peer Resources," *Proc. Peer-to-Peer Systems II,* pp. 149-159, Oct. 2003.
[21] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, "A Scalable Content-Addressable Network," *Proc. Conf. Applications, Technologies, Architectures, and Protocols for Computer Comm.,* pp. 161-172, Aug. 2001.
[22] A. Roczniak, A.E. Saddik, and P. Levy, "INCA: Qualitative Reference Framework for Incentive Mechanisms in P2P Networks," *Int'l J. Computer Applications in Technology,* vol. 29, no. 1, pp. 71-80, 2007.

[23] I. Stoica, R. Morris, D. Karger, M.F. Kaashoek, and H. Balakrishnan, "A Scalable Peer-to-Peer Lookup Service for Internet," *Proc. ACM SIGCOMM*, pp. 149-160, Aug. 2001.
[24] P. Tseng, "Convergence of Asynchronous Matrix Iterations Subject to Diagonal Dominance," LIDS technical reports, Laboratory for Information and Decision Systems, Mass. Inst. of Technology, 1988.
[25] L. Page, S. Brin, R. Motwani, and T. Winograd, "The PageRank Citation Ranking: Bringing Order to the Web," Stanford Digital Library Technologies Project, 1998.

Hiroshi Nishida received the BE degree in mechanical engineering from Kyoto University, Japan, in 1990. After working at Nintendo (Kyoto, Japan) as an electric engineer and at JGI, Inc. (Saitama, Japan) as a programmer, he studied computer science at California State Polytechnic University, Pomona, and received the MS degree in 2005. He is currently a PhD student in computer science at Oregon State University, and also works for ASUSA Corporation (Oregon) as a computer researcher. His research focuses mainly on parallel and distributed systems, especially on P2P networks. He is a member of the IEEE.

Thinh Nguyen received the BS degree from the University of Washington in 1995, and the PhD degree from the University of California, Berkeley, in 2003. He is an assistant professor at the School of Electrical Engineering and Computer Science of the Oregon State University. He has many years of experience working as an engineer for a variety of high tech companies. He has served in many technical program committees. He is an associate editor of the *IEEE Transactions on Circuits and Systems for Video Technology*, the *IEEE Transactions on Multimedia*, and the *Peer-to-Peer Networking and Applications*. His research interests include multimedia networking and processing, wireless networks, P2P networks, and network coding. He is a member of the IEEE.