Peer-Assisted Online Games with Social Reciprocity

Zhi Wang*, Chuan Wu†, Lifeng Sun*, and Shiqiang Yang*
*Tsinghua National Laboratory for Information Science and Technology
Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China
†Department of Computer Science, The University of Hong Kong, Hong Kong
wangzhi04@mails.tsinghua.edu.cn, cwu@cs.hku.hk, {sunlf,yangshq}@tsinghua.edu.cn

***Abstract*—Online games and social networks are cross-pollinating rapidly in today's Internet: Online social network sites are deploying more and more games in their systems, while online game providers are leveraging social networks to power their games. An intriguing development as it is, the operational challenge in the previous game persists, *i.e.*, the large server operational cost remains a non-negligible obstacle for deploying high-quality multi-player games. Peer-to-peer based game network design could be a rescue, only if the game players' mutual resource contribution has been fully incentivized and efficiently scheduled. Exploring the unique advantage of social network based games (social games), we advocate to utilize social reciprocities among peers with social relationships for efficient contribution incentivization and scheduling, so as to power a high-quality online game with low server cost. In this paper, social reciprocity is exploited with two give-and-take ratios at each peer: (1) peer contribution ratio (*PCR*), which evaluates the reciprocity level between a pair of social friends, and (2) system contribution ratio (*SCR*), which records the give-and-take level of the player to and from the entire network. We design efficient peer-to-peer mechanisms for game state distribution using the two ratios, where each player optimally decides which other players to seek relay help from and help in relaying game states, respectively, based on combined evaluations of their social relationship and historical reciprocity levels. Our design achieves effective incentives for resource contribution, load balancing among relay peers, as well as efficient social-aware resource scheduling. We also discuss practical implementation concerns and implement our design in a prototype online social game. Our extensive evaluations based on experiments on PlanetLab verify that high-quality large-scale social games can be achieved with conservative server costs.**

I. Introduction

Recent years have witnessed a rapid convergence of online game networks and social networks in the Internet. More and more games are deployed on social network sites, *e.g.*, Paintball [1] on Facebook, and Mafia Wars [2] on MySpace. On the other hand, the online game providers are increasingly leveraging social networks to power their games, from allowing players to make their own profiles and see game information of one another [3], to utilizing the real-world social relationships of a player as part of the game [4].

Such a new category of games are typically referred to as *social games*. Though the exact definition is still up for debate in the industry, here we consider a social game as a multi-player online game which enables real-world social connections among the players, including games on social network sites and standalone multi-player games where social connections exist among the players. Emerged as a promising industry, making social games (that are successfully popular with high-definition multimedia scenes and intensive inter-player actions), faces a significant challenge similar to that in traditional online games: A huge amount of server bandwidth is expected to be provisioned, equaling to high operational cost of the game or social network providers, *e.g.*, enabling a 3D massively multiplayer online game (MMOG) such as WoW [5] on a social network website can easily cost tens of millions of dollars per annum [6]. There have recently been proposals advocating the peer-to-peer (P2P) technology in the game network design [7], such that players (peers) directly send game updates to each other, with less dependence on the dedicated servers. Challenges remain in a peer-assisted design, among which incentivizing sufficient and stable peer bandwidth contribution has been a fundamental one. This challenge is similar to that existing in other P2P applications, but is more crucial in game update distribution given its stringent requirement on minimizing response times.

Though there have been a number of P2P incentive designs based on direct or indirect resource trading [8], [9], none of them has utilized social connections among the peers. The unique setting of a social game has made very promising a more effective *social reciprocity* based incentive for P2P game update distribution over a social network, that exploits the natural intentionality for each peer to help socially connected peers.

In this paper, we design a social game system which utilizes social reciprocity to incentivize effective bandwidth contribution and scheduling at the players, and employ peer-assisted design to distribute game updates with low server cost. Specifically, we target at 3D MMOG-like social games and investigate the distribution of player updates and interactions,[1] including their current locations, movements, voice conversation packets, *etc.*. Each player (peer) generates a "stream" of update messages over time, and distributes it to other players in its Area of Effect (AoE), *i.e.*, the area that a peer's actions can affect [10]. A player is responsible to send its own update stream to all other interested players in the AoE, either directly if it has sufficient upload bandwidth, or by resorting to *relay*

[1]We assume that the game scenes (*e.g.*, game worlds, maps) are installed before a player joins the game (*e.g.*, from a DVD), and do not consider their distribution in our system.

978-1-4577-0103-0/11/$26.00

helpers when there are too many receivers, such as the case in popular game regions [11].

Our key design is to effectively incentivize peers with extra upload bandwidth to serve as relay helpers in others' stream distribution, for which we exploit social reciprocities. We define two light-weighted give-and-take ratios at each peer: (1) peer contribution ratio (*PCR*), which evaluates the historical reciprocity level between a pair of social friends, and (2) system contribution ratio (*SCR*), that records the give-and-take level of the player to and from the entire network. A social reciprocity index (RI) is defined at each player to evaluate each other peer, based on the two give-and-take ratios and the strength of relationship between them two. This index is used in the design of two efficient algorithms for each player to optimally decide which other players to seek relay help from and help in relaying game states, respectively.

Our design is able to achieve the following effectiveness: (1) peers are maximally incentivized to contribute their available upload resources, (2) load on different relay peers is effectively balanced, and (3) upload bandwidth in the system is efficiently scheduled with social awareness (or social preference), in that players are more inclined to seek help from and help other players with closer social ties.

We have also designed detailed practical protocols to achieve our design, and implemented a prototype social game system. The implementation is extensively evaluated with experiments on PlanetLab, which validate the efficiency and effectiveness of our design in achieving high-quality large-scale social games with low server costs.

The remainder of this paper is organized as follows. We introduce the system model and the design of two give-and-take ratios in Sec. II. We present our detailed algorithm design and analysis in Sec. III. We then present extensive evaluation results based on a prototype implementation and PlanetLab experiments in Sec. IV. Finally, we discuss related work in Sec. V and conclude the paper in Sec. VI.

II. System Model

We first present the network model of our peer-assisted social game application, define social relationships in the system, and introduce the two give-and-take ratios designed to exploit social reciprocity.

A. Peer-Assisted Online Game

We consider a large-scale 3D MMOG-like game, in which players interact with each other through their controlled avatars in a virtual world. A peer (equivalently an avatar) generates a stream of continuous update messages during its game play, such as those due to walking, running, talking with other peers in the virtual world. Each peer has an AoE (Area of Effect), which covers all other avatars and objects that this peer's actions can affect, and the other avatars in the AoE should receive the update stream from the peer in a timely fashion, in order to guarantee a fluent and seamless game experience. A peer which generates an update stream is referred to as a *source peer* or a *source* in short, and a peer which receives

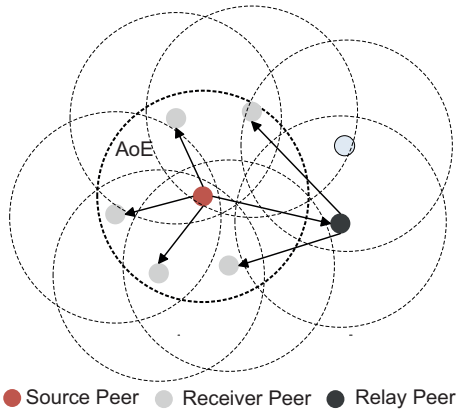

Fig. 1. System model of the peer-assisted avatar game.

the stream is referred to as a *receiver peer* or a *receiver* of the stream. In our peer-assisted game design, the source peer is responsible to distribute its update stream to all the receivers in its AoE, as it is naturally incentivized to notify others its own updates. The peer may directly send the update stream to its receivers if its upload bandwidth suffices, or ask other peers to help relay, when there are too many receivers such that its upload bandwidth is not sufficient to serve all. These helpers are referred to as *relay helpers* or *relays* in short. An illustration of the game network model is given in Fig. 1, where each circle denotes the AoE of a peer (note that the shape of each AoE could be irregular in a game, depending on the visibility of each avatar).

There are N peers in the system. Let r_i denote the average bitrate of peer i's update stream. The bitrate of update stream varies from a source peer to another, as decided by the interaction intensity of the avatar (*e.g.*, fast movement and frequent input generally lead to high bitrates); it also depends on the type of the game, *e.g.*, a game enabling voice chatting among players may generate higher-bitrate update streams than one with text messaging. Let $\mathcal{R}_i$ denote the set of receivers of peer i's update stream in its AoE and $\mathcal{H}_i$ be its set of at most K candidate relay helpers, which may be distributed across the entire game world. A relay peer may help more than one source peer concurrently. We denote the set of source peers requesting relay help from peer j as $\mathcal{S}_j$. We assume upload bandwidth at peers constitutes the bandwidth bottleneck, but download bandwidth at peers is always sufficiently large. There are a small number of dedicated servers in the system to serve as backup relay peers, which are resorted to when a source peer cannot find sufficient relay peers. A tracker server keeps track of all online avatars in the system.

The aim of our design is to effectively incentivize peers to serve as relay helpers for each other using their spare capacity, by exploring social reciprocity among peers, as well as to efficiently schedule upload bandwidth at peers, which optimizes the utilization of resources in the entire system while taking peers' social preferences into consideration. Timely distribution of update streams with minimum server involvement is our ultimate objective, which is especially important for a time-critical online gaming application.

B. Social Network Model

We assume that social relationships among the players can be organized into a social graph, where each node represents a peer, and a bidirectional edge exists between two nodes when the two peers are socially connected (*e.g.*, friends, relatives, etc.), which we refer to as *social friends* hereinafter. A weight $f_{ij} \in [0, 1]$ is associated with each edge in the social graph, denoting the strength of social connection between peer i and peer j (*e.g.*, strength of friendship). Social relationship is symmetric, *i.e.*, $f_{ij} = f_{ji}$. A larger f_{ij} represents a stronger relationship, and $f_{ij} = 0$ denotes no existing relationship.

Peers in the game system are supposed to have a *social preference*, *i.e.*, they wish to help their social friends more than the other general population. We exploit *social reciprocity* among players in our incentive design with respect to two aspects: (1) direct reciprocity between two social friends, as peers are naturally willing to help their social friends and receive their help from time to time in return; (2) indirect reciprocity among a peer and all other peers in the system, where a peer contributes resources to the system expects to receive resource contribution from others as well. Our design will take both direct and indirect reciprocities into consideration.

C. Two Give-and-Take Ratios

Two ratios are defined to evaluate the level of direct and indirect reciprocities in our system.

1) Peer Contribution Ratio (PCR): Peer contribution ratio evaluates the give-and-take balance between two social friends. PCR $W_j(i)$ is defined as the ratio of peer i's upload contribution to peer j over the total mutual contributions between the two:

$$W_j(i) = C_j(i)/(C_i(j) + C_j(i)).$$

Here, $C_j(i)$ is the total number of upload bytes that peer i has historically provided for relaying peer j's update stream, and $C_i(j)$ vice versa. $W_j(i) > \frac{1}{2}$ represents that peer i has contributed more to peer j, and $W_j(i) < \frac{1}{2}$ vice versa.

2) System Contribution Ratio (SCR): To evaluate each peer's contribution in the entire system, we also define a system contribution ratio w_i as follows:

$$w_i = y_i'/(y_i' + y_i).$$

Here y_i' is the total number of upload bytes peer i has ever provided for relaying other peers' streams, and y_i is the overall upload bytes of resources others have provided for relaying peer i's stream.

The two ratios are used in our incentive design and resource scheduling, in which peers contributing more to their social friends receive more help from the latter, and for peers with few social friends, providing more relay help to the general others will receive more help in return as well. Detailed design will be discussed in the following section.

We summarize important notations in the paper in Table I.

TABLE I
NOTATIONS

Symbol	Definition
PCR	Peer contribution ratio
SCR	System contribution ratio
RI	Social reciprocity index
y_i'	The amount of upload resource peer i has contributed to the system
y_i	The amount of upload resource others have provided to i
$C_i(j)$	The amount of upload resource peer j has provided to i
w_i	The system contribution ratio of peer i
$W_i(j)$	The peer contribution ratio of peer j between the pair of peer i and j
f_{ij}	The social closeness of peer i and peer j
$e_i(j)$	The social reciprocity index of peer j evaluated by peer i
$\mathcal{R}_i$	The receiver set of source peer i
$\mathcal{H}_i$	The relay set of source peer i
$\mathcal{S}_j$	The source set of relay peer j
u_i	The upload capacity of peer i
r_i	The stream bitrate of peer i
x_{ij}	The number of receivers that peer i has requested relay peer j to help relay to
a_{ji}	The number of receivers that relay peer j has accepted to relay for peer i
L	The number of candidate relay peers provided by the tracker server
K	The maximum number of relay peers a source maintains
T_h	The duration of a time slot

III. Detailed Design: Contribution Incentivization and Resource Scheduling

We now present our detailed incentive design for relay resource contribution, through a social reciprocity index that we define, and efficient strategies that relay peers apply to schedule their resources.

A. Social Reciprocity Index (RI)

We design a social reciprocity index $e_i(j)$ for each peer i to evaluate its perceived contribution level from another peer j, based on the two give-and-take ratios and the strength of social relationship between them two, as follows:

$$e_i(j) = (1 - f_{ij})w_j + f_{ij}W_i(j).$$

The rationale of this index is as follows: If peer j has a stronger social relationship (*i.e.*, large f_{ij}) with peer i, i evaluates j more based on relay help j has provided to itself (*i.e.*, $W_i(j)$). If little social relationship exists between peer i and peer j (*i.e.*, small f_{ij}), i evaluates j more according to relative contributions j has made to the entire system (*i.e.*, w_j).

This index is used in two effective strategies for each source to choose which other peers to seek relay help from, and for each relay to optimally decide which source peers to help, respectively. Specifically, each source will choose peers with smaller RI values it evaluates as relay helpers, and each relay tends to help sources with larger RI values it has evaluated. The idea is four-fold: (1) A source peer i prefers requesting

relay help from social friends which it has helped a lot historically (*i.e.*, smaller $W_i(j)$), or from other peers whose contribution level to the entire system is low (*i.e.*, smaller w_j); (2) A relay peer j favors helping its social friends which it has received a lot of help from historically (*i.e.*, larger $W_j(i)$), or other peers whose contribution level to the entire system is high (*i.e.*, larger w_i); (3) Social friends exchange relay help more according to the level of direct reciprocity evaluated by PCRs; (4) Peers with little social connections contribute relay resources among each other more according to SCRs, enabling multilateral reciprocity in the entire system. Detailed algorithms and rationale discussions follow in the next two sub sections.

B. Source Peer's Algorithm: Selecting Relay Helpers

When the number of receivers in the AoE of source peer i exceeds its upload capacity, i seeks relay helpers. Especially, three steps are involved: (1) source i chooses relays among a candidate set $\mathcal{H}_i$, which contains potential relay helpers with spare upload bandwidth that i has acquired from the tracker server, [2] (2) estimates how many receivers each selected relay peer may possibly help with, and then (3) assigns specific receivers to each selected relay peer.

1) Choosing Relays: Source peer i ranks all relay candidates in $\mathcal{H}_i$ in ascending order of their RIs $e_i(j), \forall j \in \mathcal{H}_i$, and chooses peers to request relay from in this order. In this way, as discussed in Sec. III-A, source i prefers social friends which it has helped a lot and asked little, or other peers which have contributed less to the system but taken more. The reason lies in that those peers are more likely to agree to help the source in relaying update streams, according to the decision algorithm to be discussed in Sec. III-C, in order to regain their give-and-take balance.

In addition, when ties occur, candidates which are also receivers in the AoE of the source peer are prioritized, as they themselves expect to receive the update stream and fewer extra relay helpers will result.

2) Estimating Relay's Available Upload Bandwidth: Each candidate relay peer, which has spare bandwidth besides sending its own game updates to its receivers, may potentially help multiple other source peers. To decide how many receivers a relay may forward source i's stream to, i carries out a probing algorithm: For new relay peer j, i randomly decides an initial number of receivers, $x_{ij}^{(0)}$, to relay j. In each following round T, if the receivers assigned to relay j have all received i's stream via j in $T-1$, i will try to assign one more receiver to relay j; otherwise, if only $a_{ji}^{(T-1)}$ receivers ($a_{ji}^{(T-1)} < x_{ij}^{(T-1)}$) are served, i will adjust $x_{ij}^{(T-1)}$ to $a_{ji}^{(T-1)}$, *i.e.*,

$$x_{ij}^{(T)} = \begin{cases} x_{ij}^{(T-1)} + 1, & \text{if } a_{ji}^{(T-1)} = x_{ij}^{(T-1)}, \\ a_{ji}^{(T-1)}, & \text{if } a_{ji}^{(T-1)} < x_{ij}^{(T-1)}, \end{cases} T = 1, 2, \ldots$$

Since a source peer only maintains at most K relay candidates, the ones with smallest estimate relay capacities will be eliminated from $\mathcal{H}_i$.

3) Assigning Receivers to Relays: We consider three different methods for sources to decide which receivers itself uploads to and which others to be served via relays, that require different levels of network delay information: (1) *Random* assignment. The receivers source i directly sends streams to and the $x_{ij}^{(T)}$ receivers to be served by relay j are randomly assigned by source i from all its receivers. No information about delay is needed. (2) *RTT-to-source* based assignment. If source i has measured its round-trip-times (RTTs) with each receiver, receivers with the largest RTTs are assigned to relay peers randomly, with the hope that the latency for them to receive the updates can be reduced by such detouring. This is based on the observation by Ly *et al.* that detouring paths can reduce delays as compared to those along original long paths [12]. (3) *Enhanced RTT-to-source* based assignment. If source i is further able to learn the RTT to each relay and the RTT between each receiver and each relay (*e.g.* by referring to a network coordinate system [13]), it assigns a receiver with large direct RTT to itself to a relay, such that the smallest end-to-end latency results. We will evaluate the performance in cases of different strategies in Sec. IV.

Our design considers only one-hop relay of update streams, *i.e.*, a relay peer receives the stream from the source peer and then forwards to receivers directly, since more relay hops may well add to delay and complexity of the game state distribution. In the case that the aggregate upload bandwidth of source i and all its relay peers is not enough to distribute the stream at rate r_i to all receivers, servers are resorted to serve as relays.

The algorithm carried out by each source peer is summarized in Algorithm 1. At source peer i, $StreamSchedule$ is called periodically. In line $01-02$, the stream is delivered to all receivers by source i directly if its upload capacity suffices. Otherwise, it resorts to the relay peers, by selecting the relay helpers that have minimum RIs (line 04), and assigning the number of receivers according to their upload capacity (line $08-09$), which is estimated using the probing strategy discussed. Notice that source i never waits for upload notification from relay peers, since the upload capacities ($x_{ij}(T)$) are updated by upload allocation ($a_{ji}^{(T-1)}$) in the previous round. When the source peer is able to address the remaining receivers, it will stop requesting relay peers (line 07); and source peer i reserves one slot of its upload capacity (equal to its streaming rate r_i), so that it can send its updates to the server for relaying when capacities from relay peers are not enough (line $07, 14, 15$).

C. Relay Peer's Algorithm: Scheduling Upload Contribution

When a player has extra upload bandwidth (beyond that used for distributing its own update streams), it may register itself as a candidate relay with the tracker server, and may take itself down from the candidate list when its upload bandwidth is fully used. The voluntary helper registration is incentivized by our upload scheduling algorithm, to be discussed next.

[2]Implementation details on how a source peer learns about receivers and candidate relay peers will be discussed in Sec. III-D.

Algorithm 1 Source Peer's Algorithm

procedure StreamSchedule()
01: **if** $u_i \geq |\mathcal{R}_i| r_i$, **then**
02: i will distribute the stream to all receivers directly
03: **else**
04: Sort $\mathcal{H}_i$ in ascending order of RIs ($e_i(j)$'s), and a relay peer that is also a receiver is prioritized
05: $u = u_i, v = |\mathcal{R}_i|$
06: **for** each relay peer j in sorted list $\mathcal{H}_i$
07: **if** $u < 2r_i$ **or** $\lfloor u/r_i \rfloor \geq v$ **then break**
08: $x_{ij}^{(T)} = x_{ij}^{(T-1)} + 1$, if $a_{ji}^{(T-1)} = x_{ij}^{(T-1)}$; otherwise $x_{ij}^{(T)} = a_{ji}^{(T-1)}$
09: Select $x_{ij}^{(T)}$ receivers from $\mathcal{R}_i$ according to assignment method
10: Send to relay j the update stream together with the list of assigned receivers
11: $v = v - x_{ij}^{(T)}$
12: **if** $j \notin \mathcal{R}_i$, **then** $u = u - r_i$
13: **end for**
14: Send the stream to remaining receivers one by one until i's remaining upload bandwidth is smaller than $2r_i$
15: Resort to the server for relaying to all remaining receivers
16: **end if**

The tracker server may provide each candidate relay peer to multiple source peers, and therefore each candidate relay can receive multiple requests from different sources simultaneously. When relay peer j's spare upload bandwidth is not enough to serve all the receivers specified in the requests, it chooses the source peers to help, prioritizing those with large social reciprocity indices ($e_j(i)$'s) it evaluates. Specifically, a relay periodically decides the source peers it helps in each time slot T, among the set of source peers $\mathcal{S}_j$, which request relay help from itself. On the other hand, once a source peer i is chosen, it is guaranteed that relay j will forward i's update stream for the duration of the time slot (T_h seconds), in order to avoid inefficiency caused by frequent relay switches.

Let $a_{ji}^{(T)}$ denote the number of receivers that relay j is to serve for source peer i in time slot T. Recall that $x_{ij}^{(T)}$ is the number of receivers source i asks relay j to serve. u_j is the maximum upload bandwidth of peer j. The source selection and upload scheduling problem is formulated into the following optimization problem:

$$\max \sum_{i \in \mathcal{S}_j} e_j(i) a_{ji}^{(T)}$$

subject to:

$$a_{ji}^{(T)} \leq x_{ij}^{(T)}, \quad \forall i \in \mathcal{S}_j,$$

$$\sum_{i \in \mathcal{S}_j} a_{ji}^{(T)} r_i \leq u_j - |\mathcal{R}_j| r_j - 2r_j,$$

$$a_{ji}^{(T)} \in \{0, 1, 2, \ldots\}, \quad \forall i \in \mathcal{S}_j.$$

Algorithm 2 Relay Peer's Algorithm

procedure AllocateUpload()
01: $u = u_j - |\mathcal{R}_j| r_j - 2r_j$
02: Sort peers in $\mathcal{S}_j$ in descending order of RIs ($e_j(i)$'s)
03: **for** each source peer i in sorted list $\mathcal{S}_j$
04: $a_{ji}^{(T)} = \min\{\lfloor u/r_i \rfloor, x_{ij}^{(T)}\}$
05: $u = u - a_{ji}^{(T)} r_i$
06: **end for**
07: Send upload allocation $a_{ji}^{(T)}$ to source peer $i, \forall i \in \mathcal{S}_j$
08: On receiving stream from source peer $i, \forall i \in \mathcal{S}_j$, forward the stream to indicated receivers

To solve this integer linear program for $a_{ji}^{(T)}$'s, we design the following heuristic: Relay peer j first allocates upload capacity of $|\mathcal{R}_j| r_j - 2r_j$ for its own stream distribution, where 2 slots are reserved for new receivers. Then j maximally allocates its spare upload bandwidth ($u_j - |\mathcal{R}_j| r_j - 2r_j$) to source peers ($i$'s) in descending order of their social reciprocity indices ($e_j(i)$'s), according to the number of receivers each source has asked j to forward streams to ($x_{ij}^{(T)}$), until its upload bandwidth becomes insufficient to forward a whole stream. In this way, social friends which have helped j a lot previously, or others which have contributed significantly to the entire system, will be given higher priority.

The algorithm carried out by each relay peer is summarized in Algorithm 2, which is invoked at the beginning of each time slot T. In line 01, the relay determines its spare upload capacity, and then allocates it to source peers according to their RIs. In our design, the relay peer reserves $2r_j$ upload capacity for itself, in case that new receiver peers may be discovered during the time slot. After that, the relay peer sends the upload allocation notification a_{ij} to all source peers in $\mathcal{S}_j$ (line 07), and the latter will adjust their receiver assignment as discussed in the source peer's algorithm. On receiving streams from source peers, the relay peer will forward the streams to the receivers (line 08).

We note that source peers are allocated relay resources according to how much help they have provided historically, as captured in $e_j(i)$. This incentivizes a peer with spare upload resource to contribute relay help (by registering with the tracker server), such that when it needs relay help for distributing its own updates, its social reciprocity indices can rank high at other peers, and it can easily receive relay help in return. More discussions on effectiveness of our design will be given in Sec. III-E.

D. Implementation Discussions

We next discuss key implementation issues of our design in practice.

1) Maintenance of Give-and-Take Ratios: The amount of resources a peer i has contributed to/taken from its social friends and the entire system, *i.e.*, $C_j(i)$'s, $C_i(j)$'s, y_i', and y_i, are recorded since peer i registered an account with the online game system. In our implementation, mutual resource

contributions between peer i and peer j ($C_j(i)$ and $C_i(j)$) are maintained at both i and j, while system contributions of each peer i (y_i' and y_i) are maintained by the tracker server, similar to that in the existing private P2P file sharing systems [14]. In particular, after relay j helps source i in serving a_{ji} receivers for Δt seconds, $C_i(j)$ maintained at both peer i and peer j will be increased by the total number of upload bytes $a_{ji}r_i\Delta t$, while y_i and y_j' maintained at the tracker server will be increased by $a_{ji}r_i\Delta t$ as well.

2) Receiver and Relay Discovery: In our design, a source peer actively discovers its receiver peers and relay helpers. For receiver discovery, the virtual world is divided into small regions, whose size is at the same magnitude as players' AoEs. Each peer periodically reports its current location to the tracker server. A source peer learns about receivers in its AoE via the tracker server.

When a source peer needs (more) relay peers, it contacts the tracker server, which will provide L peers randomly selected from its candidate relay pool. In our implementation, a peer j with extra upload capacity ($\tilde{u}_j = u_j - |\mathcal{R}_j|r_j \geq 2r_j$) may register as a relay helper. When its extra upload capacity is below $2r_j$, it will no longer serve as relay helpers. At the source peer, it keeps the size of relay helper set $\mathcal{H}_i$ no larger than the maximum relay number of K, and candidate relay peers with the smallest upload allocation to i will be eliminated when new relay peers are discovered from the tracker.

3) Relay Durations: We have designed that a relay j will serve a receiver for at least the duration of one time slot, *i.e.*, T_h seconds. During each time slot, new relay requests from source peers may arrive from time to time, which will be inserted into set $\mathcal{S}_j$; peer j will decide its new upload bandwidth allocation at the beginning of the next time slot. The choice of T_h renders a tradeoff: small T_h may lead to frequent relay switches for source peers, while large T_h may result in less efficient utilization of relay peers' upload bandwidth, as relay requests from source peers with high priority may not be timely addressed. We will evaluate the effects of different T_h values in our experiments.

Note that although we have introduced "rounds" at peers for execution of source and relay algorithms, our system operates in a fully asynchronous fashion: individual peers carry out the designed protocols periodically, while rounds at different peers do not need to be synchronized at all.

E. Analysis of Design Effectiveness

We analyze our design, and show that it achieves the three objectives listed in the introduction, namely peers' maximal upload contribution, load balancing among relays, and efficient upload bandwidth scheduling with preference towards close social ties.

1) Incentives for Upload Contribution: In our design, relay peers allocate their upload bandwidth to requesting source peers in descending order of their social reciprocity indices (RIs) that it calculates. The more help source peer i has provided to relay peer j or to the entire system, the larger RI ($e_j(i)$) j will evaluate towards i, and the more likely j will help i in relaying its update streams. Source peer i does not have information about RIs that relay j evaluates towards other requesting sources, and has no idea whether its own RI is large enough to be selected by j. Therefore, peer i has to always keep its RI at a high level, by maximally serving all its own receivers directly (to prevent decrease of RIs due to seeking relay help), and by contributing spare upload resource whenever there is, in order to enhance its system contribution ratio and peer contribution ratios to others (to boost increase of its RIs). Consequently, our design effectively incentives upload bandwidth contributions at all peers in the game system. Since some peers may have quite small RIs due to their low upload capacities, it could be difficult for them to obtain enough relay help. A promising approach is that these peers can buy extra game "points" from the game provider or operator, and pay the server or other peers to relay updates for them. We will address this mechanism in future work.

2) Load Balancing at Relays: In source peer's algorithm in Sec. III-B1, a source peer i chooses relays from all candidates in ascending order of their RIs, $e_i(j) = (1 - f_{ij})\frac{y_j'}{y_j'+y_j} + f_{ij}\frac{C_i(j)}{C_j(i)+C_i(j)}, \forall j \in \mathcal{H}_i$. Relays with smaller RIs, *i.e.*, which source i has helped a lot but asked little from, or that have contributed less to the system but taken more from, are more likely to be chosen. If relay j is chosen and forwards i's streams, y_j' and $C_i(j)$ will increase, then $e_i(j)$ will increase, and peer j's rank lowers in source i's relay selection. In the next round, relay j may not be selected by source i — other candidate relays with smaller contributions will be chosen — until its contribution level becomes relatively low again. In this way, our scheme enables balanced resource utilization at all peers, by always having source peers use relays with historical lowest contribution levels.

3) Social Preference: Besides historical contributions, social closeness f_{ij} between peers is also considered in evaluating RIs. In relay peer's algorithm in Sec. III-C, a relay peer j decides sources to help according to descending order of their RIs, $e_j(i) = (1 - f_{ij})\frac{y_i'}{y_i'+y_i} + f_{ij}\frac{C_j(i)}{C_i(j)+C_j(i)}, \forall i \in \mathcal{S}_j$. A source i with closer social relationship (larger f_{ij}) and more historical contribution to j (larger $\frac{C_j(i)}{C_i(j)}$) is more likely to derive a larger $e_j(i)$ and thus has higher priority to be chosen by relay j. Therefore, mutual relay help between peers with close social ties is facilitated, while upload bandwidth in the entire system is efficiently scheduled.

IV. Performance Evaluation

We evaluate the performance of our design based on a prototype implementation of a social game system and extensive experiments on PlanetLab.

A. Prototype Implementation and Experimental Settings

We implement a prototype multi-player game in C++ programming language, where avatars interact and move around in a virtual world. The world is divided into 15×15 regions, each of which is of size 40×40. 240 PlanetLab nodes are used in our experiments, corresponding to the total number

of avatars. Avatars are distributed initially across the game regions following a *zipf* distribution.

The mobility pattern of each avatar is as follows: an avatar moves across these regions, and stays in one region for a short period. The time an avatar spends in a particular region is proportional to the region's popularity, which varies from 1 second to 300 seconds; when an avatar moves, its moving speed is randomly chosen from the range of $[5, 8]$; and the probability that it moves towards a specific direction (*i.e.*, a specific adjacent region) is proportional to the popularity of that adjacent region. Avatars do not move back to the same region within 10 seconds. The radius of an avatar's AoE is 20. Each avatar is generating a stream of update packets at the rate of 80Kbps. Each packet for update stream has the size of 1KB. We limit the upload bandwidth of the peers (download bandwidth is never the bottleneck), such that 20% of the nodes have an upload capacity of 512Kbps, 60% have upload bandwidth of 1Mbps, and the rest 20% are 2Mbps.

In our prototype system, there is a tracker server, helping each source peer find receivers and relays. There is also a dedicated server to serve as a backup relay. In our default settings, the number of candidate relay peers supplied by the tracker server per inquiry, L, is 20, time slot duration T_h is 3 seconds, and the maximum number of relay candidates per source, K, is 10. Packets of update streams and other control messages are delivered over UDP. Each packet is stamped with the time t_1 when it is sent out from the source, and also the time t_2 when it is received by the receiver. In order to derive the end-to-end delay $t_2 - t_1$ of packet distribution, we synchronize clocks at all nodes (but note that synchronization is not needed for carrying out our algorithms). Each of our experiments lasts about 40 minutes.

Social closeness f_{ij} between peers are set following a distribution summarized based on a set of runtime traces from Renren [15], one of the largest online social network sites in China. The traces we obtained from Renren operators contain information of more than $10,000$ users and their friend lists. We derive the social closeness f_{ij} between peer i and j in the following intuitive fashion: We suppose peers sharing more common friends have closer relationship than those with fewer common friends, and evaluate f_{ij} as the fraction of common friends of peer i and j, $f_{ij} = \frac{|F_i \cap F_j|}{|F_i \cup F_j|}$, where F_i denotes the set of friends of peer i.

B. Effectiveness of Contribution Incentivization

We investigate the effectiveness of our incentives by evaluating a success ratio of relay requests, issued by different source peers. In our design, the relay request sent from a source peer to a relay may not be served, as relays carry out a source selection algorithm when its spare upload bandwidth cannot serve all the relay requests. If a relay request is not served, the source has to request relay help again from another relay, which introduces additional delay into its stream distribution. Therefore, the request success ratio of a source peer i, defined as the fraction of the number of its relay requests that are fully served by corresponding relays (*i.e.*, at relays $j \in \mathcal{H}_i$ where $a_{ji}^{(T)} = x_{ij}^{(T)}$), over the total number of relay requests it has issued over time, can be used to represent distribution efficiency of a peer's update stream. Fig. 2 illustrates the request success ratios of source peers against their respective average upload bandwidth contributed for relaying others' streams over the duration of the experiment. Each sample in the figure represents one peer in the system. We observe that peers with higher upload contribution can obtain larger request success ratio when they need relay help to distribute their own update streams. In this way, peers are incentivized to contribute more upload bandwidth according to our design.

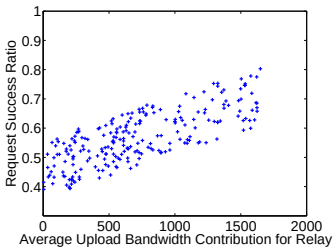

Fig. 2. Request success ratio vs. upload bandwidth contribution for relay at all peers.

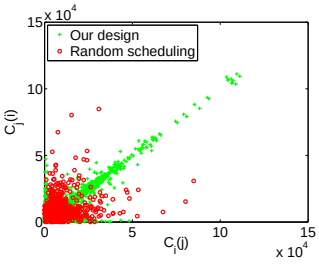

Fig. 3. Mutual contribution between peer pairs.

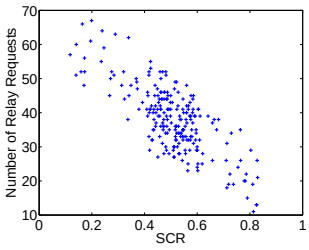

Fig. 4. Relay request load vs. system contribution ratio at all relay peers.

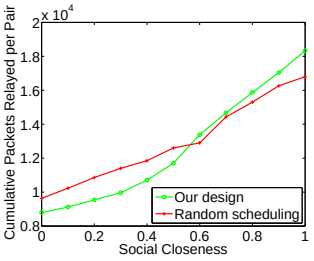

Fig. 5. Cumulative number of packets relayed per source-relay pair over levels of social closeness between the source-relay pair.

C. Load Balancing on Peers

We first study mutual relay help between peer pairs. We compare our design with a simple *random scheduling scheme*, in which source peers randomly select relays from its candidate pool and relay peers randomly choose sources to help, respectively. Fig. 3 plots the overall number of bytes peer i has helped relay for peer j against the total number of bytes peer j has uploaded for relaying peer i's stream throughout the duration of the experiment, for all pairs of peers in the system. Each sample in the figure represents one peer pair. We observe that mutual resource contribution between two peers is much more balanced with our design than that in the random scheduling scheme.

Fig. 4 illustrates the load of relay requests at each relay peer against its system contribution ratio. The load of relay requests is calculated as the number of requests received by each relay peer in the last 5 minutes, against the system contribution ratio at the beginning of this period. Each sample represents one

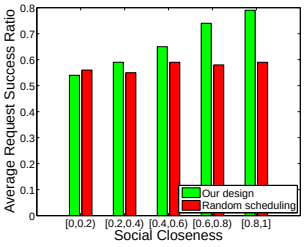

Fig. 6. Average request success ratio vs. social closeness among peer pairs.

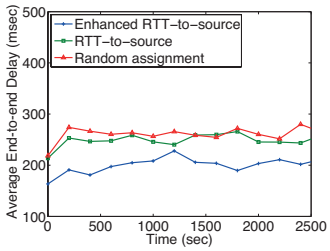

Fig. 7. End-to-end stream distribution delays: a comparison among three receiver-to-relay assignment methods.

relay peer. Relay peers with larger SCRs (more contribution to the system) receive less requests than peers with smaller SCRs (less contribution to the system), validating the effectiveness of load balancing among relay peers with our design.

D. Social Preference

Fig. 5 investigates the impact of social closeness between sources and relays on the upload resource allocation at the relays. We categorize source-relay pairs according to the social closeness between the two (*i.e.*, f_{ij}), and count in each category the number of packets each relay has forwarded for the corresponding source throughout the duration of the experiment. This number is further divided by the number of source-relay pairs in each category to generate the number of packets relayed per source-relay pair, and plotted against the social closeness level of the category in a cumulative function fashion in Fig. 5. We can observe that our design achieves better social preference, in that relay help is exchanged more between social friends than that in the random scheduling scheme. The reason lies in that a pair of social friends may provide relay help to each other for many times throughout their game play, according to the source and relay selection strategies in our design.

To further illustrate social preference achieved by our design, we divide all pairs of peers into 5 groups according to their social closeness range, and calculate the average request success ratio (defined in Sec. IV-B) between peer pairs in each group, and plot the results in Fig. 6. The results again show that our design achieves better social preference than the random scheduling scheme, in that the average request success ratio is larger when social ties between peer pairs are stronger with our design, while it does not change much with the variation of social closeness levels in the random scheduling scheme.

E. Stream Distribution Delay

We evaluate the average stream distribution delay at each source peer, calculated as the average end-to-end latency for each of its update packets to be delivered from the source to the receivers. Fig. 7 compares the average end-to-end stream distribution delay among all source peers, when different receiver-to-relay assignment strategies are applied, as discussed in Sec. III-B3. As expected, the enhanced RTT-to-source algorithm leads to the lowest stream distribution latency, when information of RTTs between peers is available. Otherwise, random assignment achieves similar delay performance to the RTT-to-source based assignment, while incurring less RTT measurement overhead. In our other experiments, we have used random assignment for receiver-to-relay mappings as the default.

F. Server Load and Protocol Overhead

In Fig. 8 and 9, we study the relay load on dedicated servers and the protocol overhead in our system, in terms of different maximum number of relays each source maintains, K, and the length of a time slot, T_h. The server load is defined as the fraction of update packets that are relayed by the server over all packets distributed in the system throughout the duration of the experiment. The overhead in our algorithms includes control messages sent by source peers to discover receivers and relays, and those used by relay peers to register with the tracker server and allocate its upload bandwidth, etc. We evaluate the control overhead using the ratio of the total number of bytes in control messages over the total number of bytes of game updates distributed in the system.

In Fig. 8, we can see that if T_h is too large or too small, server load is higher. The reason is that for larger T_h, more relay resource is wasted as being locked for a source peer which no longer needs it. While for smaller T_h, the relay needs to reschedule the requests to serve frequently. With respect to control overhead, Fig. 9 shows that smaller T_h leads to larger overhead, since more control messages are sent among relay peers and source peers for relay scheduling.

On the other hand, we observe that a larger K leads to lower server load, since source peers can select among a larger set of relay candidates, and their requests are more likely to be fully served. However, larger K also leads to higher control overhead, since more messages are sent to maintain these many relays at each source peer.

Based on these observations, we have used the default values of $T_h = 3$ seconds and $K = 10$ in our other experiments, which achieve a good tradeoff between server load reduction and control overhead.

The above results also show that the bandwidth needed to send control messages for maintaining our P2P system is quite low, as compared to the streaming bandwidth demand. In addition, the load on dedicated servers can been significantly alleviated by using our P2P incentive mechanism, to implement a high-quality social game system.

V. Related Work

Incentive engineering has been an important issue of P2P networks since its inception [16]. Different strategies have been designed to encourage peers' mutual sharing of upload resources, *e.g.*, tit-for-tat used in Bittorrent [17], reputation systems [18], and other game-theoretical approaches [19]. Recently, Liu *et al.* [20] have observed that better sharing performance can be achieved in some private P2P file sharing systems, by applying an admission control mechanism based on upload/download ratio of each peer.

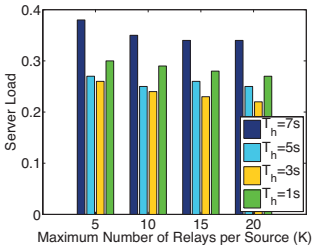

Fig. 8. Server load.

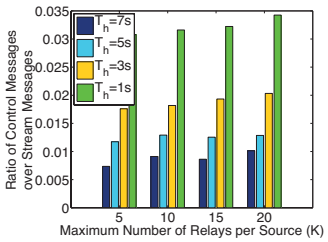

Fig. 9. Protocol overhead.

A few recent studies have explored social connections among nodes for better content distribution in a network. Pouwelse *et al.* [21] design a social network based file sharing system, which enables fast content discovery and recommendation. Liu *et al.* [22] design a network-wide tit-for-tat scheme, where peers can trade resources via a social chain in a P2P network. Li *et al.* [23] enable social preference during message routing in a delay tolerant network, where nodes are more likely to route messages for those with social relationships.

Peer-assisted online game design has been proposed in recent years. Pittman *et al.* [24] have measured the popularities of regions in the virtual world in the MMOGs, which are shown to follow a *power-low* distribution. Bharambe *et al.* [25] propose Donnybrook, a system that enables First-Person Shooter (FPS) games without dedicated server resources. Varvello *et al.* [7] design and evaluate a P2P communication infrastructure to distribute the management of the virtual world in Second Life, based on the structured P2P network Kad. Hu *et al.* [26] propose FLoD, in which the 3D contents in the virtual environment are delivered as media streams by peers.

VI. Concluding Remarks

This paper advocates to utilize social reciprocities among peers for efficient contribution incentivization and scheduling, in order to power high-quality online games with low server cost. We exploit social reciprocity with two light-weighted give-and-take ratios at each peer, which record peer's contributions to social friends and to the entire system, respectively. We also design efficient peer-to-peer mechanisms for game state distribution based on a combination of peers' social relationship and historical contribution levels. Through analysis and extensive experiments using a prototype implementation on PlanetLab, we show that our design is able to achieve effective incentives for resource contribution, load balancing among relay peers, as well as efficient social-aware resource scheduling. All these verify that high-quality large-scale social games can be achieved based on our design. For future work, we are going to evaluate our strategy in more realistic games, and design mechanisms to avoid attacks such as cheating of peers.

Acknowledgment

This study is supported in part by National High-Tech Research and Development Plan of China (863) under Grant No. 2009AA01Z328, the National Natural Science Foundation of China under Grant No. 60833009 and No. 60773158, the National Basic Research Program of China (973) under Grant No. 2006CB303103, and the Research Grants Council of Hong Kong under contract No. HKU 718710E.

References

[1] http://paradisepaintball.cmune.com.
[2] http://www.myspace.com/.
[3] *Eve Online Game Introduces SpaceBook Social Network*, http://www.socialtimes.com/2010/05/eve-online-game-introduces-spacebook-social-network.
[4] *Parking Wars*, http://www.aetv.com/parking-wars/index.jsp.
[5] "http://www.worldofwarcraft.com."
[6] "http://www.wired.com/gamelife/2008/09/total-operating/."
[7] M. Varvello, C. Diot, and E. Biersack, "P2P Second Life: Experimental Validation Using Kad," in *Proc. of IEEE INFOCOM*, 2009.
[8] M. Feldman and J. Chuang, "Overcoming Free-riding Behavior in Peer-to-Peer Systems," *ACM SIGecom Exchanges*, vol. 5, no. 4, pp. 41–50, 2005.
[9] L. Jian and J. MacKie-Mason, "Why share in peer-to-peer networks?" in *Proc. of the 10th International Conference on Electronic Commerce*, 2008.
[10] R. Sueselbeck, G. Schiele, S. Seitz, and C. Becker, "Adaptive Update Propagation for Low-latency Massively Multi-user Virtual Environments," in *Proc. of IEEE Computer Communications and Networks (ICCCN'09)*, 2009.
[11] B. Knutsson, H. Lu, W. Xu, and B. Hopkins, "Peer-to-peer Support for Massively Multiplayer Games," in *Proc. of IEEE INFOCOM*, 2004.
[12] C. Ly, C. Hsu, and M. Hefeeda, "Improving Online Gaming Quality using Detour Paths," in *Proc. of ACM Multimedia*, 2010.
[13] F. Dabek, R. Cox, F. Kaashoek, and R. Morris, "Vivaldi: A Decentralized Network Coordinate System," in *Proc. of the 2004 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, 2004.
[14] C. Zhang, P. Dhungel, Z. Liu, and K. Ross, "Bittorrent Darknets," in *Proc. of IEEE INFOCOM*, 2010.
[15] *Renren*, http://www.renren.com.
[16] M. Zghaibeh and K. Anagnostakis, "On the Impact of P2P Incentive Mechanisms on User Behavior," *NetEcon+ IBC*, 2007.
[17] B. Cohen, "Incentives build robustness in BitTorrent," in *Proc. of Workshop on Economics of Peer-to-Peer systems*, 2003.
[18] S. Marti and H. Garcia-Molina, "Taxonomy of Trust: Categorizing P2P Reputation Systems," *Computer Networks*, vol. 50, no. 4, pp. 472–484, 2006.
[19] R. Ma, S. Lee, J. Lui, and D. Yau, "A Game Theoretic Approach to Provide Incentive and Service Differentiation in P2P Networks," in *Proc. of ACM SIGMETRICS*, 2004.
[20] Z. Liu, P. Dhungel, D. Wu, C. Zhang, and K. Ross, "Understanding and Improving Ratio Incentives in Private Communities," in *Proc. of IEEE ICDCS*, 2010.
[21] J. Pouwelse, P. Garbacki, J. Wang, A. Bakker, J. Yang, A. Iosup, D. Epema, M. Reinders, M. V. Steen, and H. Sips, "Tribler: A Social-based Peer-to-Peer System," *Concurrency and Computation: Practice and Experience*, vol. 20, no. 2, pp. 127–138, 2008.
[22] Z. Liu, H. Hu, Y. Liu, K. Ross, Y. Wang, and M. Mobius, "P2P Trading in Social Networks: The Value of Staying Connected," in *Proc. of IEEE INFOCOM*, 2010.
[23] Q. Li, S. Zhu, and G. Cao, "Routing in Socially Selfish Delay Tolerant Networks," in *Proc. of IEEE INFOCOM*, 2010.
[24] D. Pittman and C. GauthierDickey, "A Measurement Study of Virtual Populations in Massively Multiplayer Online Games," in *Proc. of the 6th ACM SIGCOMM Workshop on Network and System Support for Games*, 2007.
[25] A. Bharambe, J. Douceur, J. Lorch, T. Moscibroda, J. Pang, S. Seshan, and X. Zhuang, "Donnybrook: Enabling large-scale, high-speed, peer-to-peer games," in *Proc. of ACM SIGCOMM*, 2008.
[26] S. Hu, T. Huang, S. Chang, W. Sung, J. Jiang, and B. Chen, "Flod: A Framework for Peer-to-Peer 3D Streaming," in *Proc. of INFOCOM*, 2008.