Tribler-G: A Decentralized Social Network for Playing Chess Online

Egbert Bouman, Alexandru Iosup, and Dick Epema
Department of Electrical Eng., Mathematics and Computer Science
Delft University of Technology, Delft, The Netherlands
E.Bouman@gmail.com,{A.Iosup,D.H.J.Epema}@tudelft.nl

Abstract Online board games such as online chess, which can be played through specialized servers or as add-ons to social networking sites, entertain millions of players. However, these servers and sites are essentially centralized, and thus have drawbacks such as high hosting costs, limited scalability, and single points of failure. As an alternative, in this work we have designed GameCast, a decentralized protocol for playing turn-based multi-player board games. Using GameCast, communities of gamers form peer-to-peer overlay networks that enable playing and commenting on games, and alleviate the cost, availability, and scalability problems of centralized servers. The GameCast protocol supports three operations: disseminating information about players and games, reaching agreement on starting a game, and playing a game. We have implemented GameCast as Tribler-G, an extension to the Tribler peer-to-peer file-sharing system. To evaluate the performance of Tribler-G, we have conducted real-world experiments with hundreds of peers, the results of which show that GameCast functions properly and scales well.

1 Introduction

Despite the availability of many sophisticated online games which use complex graphics and animations, traditional board games remain popular in the online world, as evidenced by the large numbers of players they attract on social networking sites. The popular systems rely on centralized architectures, which have inherent cost, scalability, and availability drawbacks. Additionally, many current systems are commercially oriented, and their approach to generate revenue (for example, through advertisements) may hinder the gaming experience. As an alternative, in this paper we design, implement, and evaluate GameCast, a decentralized protocol for building and maintaining online social networks of people who play board games.

Centralized alternatives already exist for users who wish to play board games online: social networking sites, such as Facebook; general portals, such as Yahoo! through its Games lounges; and servers dedicated to a particular game, such as the (paid) Bridge Base Online servers, the (paid) Internet Chess Club (ICC) server, and the Free Internet Chess Server (FICS). FICS, due to its good-quality and free service, is popular: over 20 million games were played by its over 350,000 registered members in 2011. However, these centralized systems may not be able to support an important increase of the number of players, triggered, for example, by the increasing popularity of mobile platforms. When the number of concurrent users is large, centralized approaches lead to significant hosting

costs, to limited scalability, and reduced availability. Currently, the standard practice of game operators is to purchase and maintain over-provisioned infrastructures, which consume up to 40% of the total game revenues in a yearly market of over 24 billion dollars. For example, World of Warcraft (WoW)—a non-board online game with millions of users—consumes over $50 million per year for maintenance [1]. Scalability is limited for the online games which apply admission control techniques [2,3] to maintain good quality of service. WoW allows at most 5,000 concurrent players for each of its over 400 server clusters; because players are not allowed to switch between server clusters, waiting occurs when any server cluster is overloaded. The availability of online gaming servers is still an important concern in the industry; for example, WoW servers are taken offline more than four hours each week.

In this work we propose a decentralized approach to operating online board games. Previous work on decentralized online gaming has focused on other types of games, such as first-person shooters and massively multiplayer role-playing games. Addressing the complexity of the game world, the primary concerns of previous research are partitioning [4,5,6] and timely delivery [7,8] of game state. In contrast, in this paper we design GameCast, a decentralized, peer-to-peer-based protocol that allows building social networks of online game players, including finding suitable opponents, managing ratings, and discussing games. GameCast uses a gossip protocol for disseminating information, and is in terms of message types and structure compatible with the centralized Internet Chess Server protocol used by FICS, ICC, and other online chess servers. In addition, we provide Tribler-G, an implementation of GameCast that is based on the Tribler peer-to-peer file-sharing system [9]. Tribler-G builds a gaming network overlay, independent from Tribler's own, for communities of board-game players. The main contribution of this work is twofold:

1. We design the GameCast protocol that enables a decentralized social network for playing board games online, and the Tribler-G system that implements GameCast for chess (Section 3).
2. We evaluate the Tribler-G system in real-world scenarios, when supporting hundreds of players on our DAS-4 multi-cluster, and show evidence of good functionality and scalability (Section 4).

2 Background

In this section we present two third-party systems used in this work.

2.1 Internet Chess Servers

A popular way to play chess online is through the use of an Internet chess server (ICS). The communication protocol between an ICS and a player is a text-based adaptation of the `telnet` protocol; this allows standard telnet and graphical clients, such as xboard and Winboard, to play through the ICS. The ICS protocol enables users to invite one another and to play games of chess. The standard ICS protocol, albeit designed originally only for playing chess, has an extensible syntax that already supports a variety of board games.

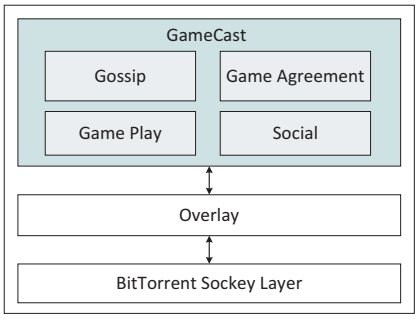

Figure 1. The software architecture of each peer in Tribler-G.

The most popular ICSs have substantial user-bases. For example, the Free Internet Chess Server (FICS) has over 350,000 registered users and at least 900 users are logged in at any time. To engage these large communities in our peer-to-peer system, we design its core GameCast protocol to support the syntax of the ICS protocol (see Section 3.2).

2.2 Tribler

Tribler [9] is a file-sharing application based on the BitTorrent protocol. Tribler extends the BitTorrent protocol; among the new features, Tribler introduces the notion of the overlay swarm, a virtual swarm encompassing all peers that are using the Tribler software. The overlay swarm is secured through public-key cryptography—each peer has a public/private key-pair with which peers in the overlay swarm can authenticate each other.

In Tribler, each peer downloading files accumulates a history of downloads, essentially a preference list, which they can exchange through the overlay swarm via Tribler's BuddyCast algorithm. Peers with similar preference lists are *taste buddies*. BuddyCast utilizes an epidemic protocol [10] to discover taste buddies, and ensures that taste buddies periodically connect and exchange preference lists. To improve the discovery of taste buddies, BuddyCast also employs the opportunistic mechanism of exchanging preference lists among random peers.

3 The Design of Tribler-G

In this section we introduce Tribler-G, a system for playing online chess and socializing over a peer-to-peer network.

3.1 Tribler-G Design Overview

Tribler-G is a peer-to-peer system with the functionality of an online social gaming network. Each peer in Tribler-G runs the software architecture depicted in Figure 1. The core of Tribler-G is GameCast, our peer-to-peer protocol for operating a social network of game players. Message exchange between peers occurs via the BitTorrent Socket Layer, which accepts incoming TCP connections and hands them to the Overlay; in turn, the Overlay sends only authenticated messages to GameCast. By leveraging a peer-to-peer protocol, Tribler-G alleviates the cost, availability, and scalability issues of centralized game servers: the cost

Message type	Function
gossip	Exchanges peer and game information.
seek	Notifies peers of a random peer invite.
match	Notifies a particular peer of a personal invite.
play	Responds to a random peer invite.
accept	Accepts either a play or a match message.
decline	Declines either a play or a match message.
unseek	Notifies peers that a random peer invite is no longer available.
start	Tells a player that the game has begun.
abort	Offers the opponent to abort.
draw	Offers the opponent a draw.
resign	Resigns the game. The opponent is declared the winner.
move	Notifies a peer of a move.
discuss	Attaches a comment to a game

Table 1. The GameCast message types and their functions, grouped by category.

of sharing is shared among all peers; failures of game servers affect only the peers involved in the game instance; the operational capacity of the system *increases* with the addition of new players.

To operate, Tribler-G maintains an overlay network, separate from the ordinary Tribler overlay [9], that allows for the creation of social gaming networks. Using this overlay network, users are able to create new games, to join existing games, and to play games via the GameCast protocol. Once a game is finished, it will be distributed by GameCast throughout the network so that other players can comment on the game. GameCast supports four main types of communication, related to social interaction (Section 3.2), general information exchange (gossiping, see Section 3.3), game agreement (Section 3.4), and gameplay (Section 3.5). Tribler-G gives players access to ratings of other players, and also supports, through the design of GameCast message syntax (Section 3.2), playing games against users on the centralized FICS server.

Although GameCast supports various types of board games, including those with more than two players, Tribler-G currently only implements online chess. For player rating, which is used during game agreement (Section 3.4), Tribler-G supports the Glicko [11] *rating system*. The same rating system, which only supports two-person games, is used by FICS. Player ratings are calculated by each peer independently, using only local game information; because rating updates occur only when games are finished, this approach provides a good rating estimate for slow-paced games such as chess.

3.2 GameCast Messaging

GameCast *messages* contain simple text-based strings, formatted to identify the message type and a number of message-specific arguments. Wherever possible, we use the same message syntax as the ICS protocol (see Section 2.1). Several GameCast messages include information about games, which consists of at least the type of game, identifiers of the peers that play the game, the type of game clock, and any number of game-specific parameters. For finished games, messages also include information about the moves made, the outcome of the game, the scores of each player, and the winning and other per-side status.

The GameCast protocol uses several *message types* (shown in Table 1), which are grouped in four categories (separated in the table by horizontal lines). `Gossip` messages are used in the dissemination of peer and game information. `Seek`, `match`, `play`, `accept`, `decline`, and `unseek` messages are used for reaching game agreements. `Start`, `abort`, `draw`, `resign`, and `move` messages are used for enabling game play. Finally, `discuss` messages provide people on the network with the ability to attach comments to games; to attach a comment to a game, peers send a `discuss` message to the owner, who will then spread the comment through the exchange of future `gossip` messages (see Section 3.3).

3.3 GameCast Information Dissemination

GameCast uses a gossip protocol to disseminate game information across the network. Analogous to BuddyCast (see Section 2.2), GameCast discovers peers within the network and forms an overlay network. The resulting overlay network, which we call the *game network*, is constructed such that peers that frequently play games against each other tend to cluster. This network structure is based on the idea that users who play against each other frequently will likely also be interested in each others' games.

To achieve this network structure, we use the notion of *game buddies*, that is, peers that play games against each other frequently. To quantify play frequency, we define the *interaction factor* of peer i and j as $\min(f_{ij}, c)/c$, where f_{ij} denotes the number of games peers i and j have played against each other, and c is a threshold (e.g., 50) above which the play activity between two players is considered to be frequent. Connected peers with the highest interaction factors are called game buddies. Each peer on the game network maintains connections to at most 10 game buddies and 10 randomly chosen peers.

Information is spread within the network through the exchange of gossip messages. When a peer decides to send such a message, it needs to select a target peer. The type of the target peer alternates between a game buddy and a (uniformly) random peer within the game network. The reason for introducing random peers is to explore new peers (and games) in the network. When choosing a game buddy target, the peer with the highest known interaction factor is chosen. The receiving peer will update its database with the information from the message, and will then send a gossip message back. After each exchange, the target peer is excluded from consideration for a pre-defined amount of time, to avoid exchanging information with the same peer too frequently.

To provide newly arrived peers with an initial list of peers (i.e., bootstrapping), we use *bootstrapping peers*, which are peers that are always online, but do not take part in the game agreement process or play games. A list of bootstrapping-peers is included with each Tribler-G installation.

3.4 GameCast Game Agreement

To play a game, peers need to contact each other and to agree on starting the game. This it done through the process of game agreement, which we explain in the following.

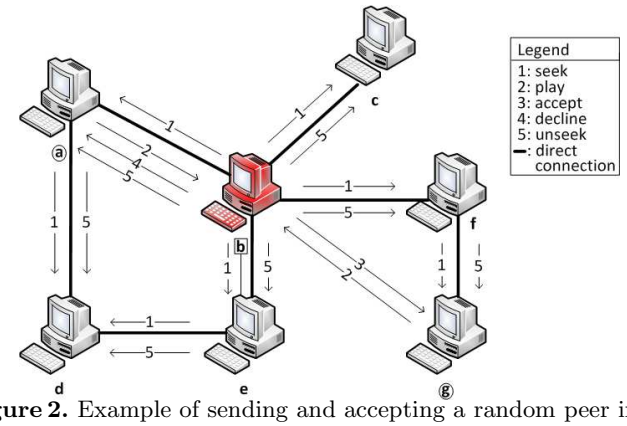

Figure 2. Example of sending and accepting a random peer invite.

Each game has an owner associated with it, which identifies the peer who created the game. The *owner is responsible* for finding the appropriate number of players that are required to start the game; this is achieved through invitation-related messages (*invites*). Additionally, the owner of a game sets the parameters of the game (e.g., what color to play with, the maximum time the game should take), which are included when an invite is sent. GameCast currently supports two types of invites. A *personal invite* is sent when one peer (the inviter) wants to invite a *specific* set of other peer(s) (the invitees) for a game. To this end, the inviter sends a `match` message (see Table 1) to each invitee. When the invitee receives the `match` message, it will respond with an `accept` or a `decline` message.

A *random peer invite* is sent when a peer wants to invite *any* peer(s) with the play rating within a certain range. An example of the random peer invite process for a two-player game is depicted in Figure 2. The inviter (peer b) sends a `seek` message to all connected game buddies and random peers (peers a, c, e, and f). The peers receiving the `seek` message forward it to their game buddies and neighboring random peers (peers d and g), until the message is distributed over a distance of two hops. Peers who decide to accept the invitation (peers a and g) will respond to the invite by sending a **play** message back to the inviter. When the inviter receives the first `play` message, it accepts it through an `accept` message sent to the corresponding peer. The inviter declines all further `play` messages related to the invite via `decline` messages. For a two-player game, the invite has become invalid, so the peers in the network need to be notified. To this end, the inviter sends an `unseek` message in the same way the `seek` message was sent.

3.5 GameCast Gameplay Enactment

The gameplay enactment starts immediately after a successful game agreement: the owner sends a `start` message to all the players who agreed to participate in this game instance. The `start` message includes the complete list of players, allowing each player to be aware of the current game participants (*current players*). Each player who receives a `start` message assumes that the game has started and marks the time.

Currently, GameCast supports only games that have a predetermined order in which players make their moves. This means that, since the identifiers of the players have already been established during the invitation process, each player is now independently able to determine when it is time to move. Players move by sending a `move` message to all current players. Peers receiving a `move` message check the validity of the move, the order of the move (see below), and whether the move was made in time (see also below). If all these verifications are successful, the peer imports the received move message into its game state database.

A game is finished when a time-out has occurred or a closing move (i.e., a move after which the game will be finished) is received. Current players observing the end of their game stop participating in the game; additionally, the owner will start distributing the game using `gossip` messages (see Section 3.3).

For a game with more than two players, *move ordering* has to be correctly ensured, as it is possible that `move` messages are received in a wrong order. To counter this situation, peers number their `move` messages consecutively, and buffer `move` messages that are received out-of-order until they are ready to be processed. The size of this buffer can be limited to the number of players within the game instance, since peers should receive exactly one `move` message from every other peer between two moves made by themselves.

Many competitive games such as chess, go, and Scrabble utilize *game clocks* to keep track of the total time each player has used. Similarly to FICS (see Section 2.1), GameCast uses Fischer-after clocks. Fischer-after clocks count down from the initial number of minutes, and add to the clock of a player a predefined amount of time after the player moves. The difference between the time at which a player makes a move and the moment the other player(s) receive the corresponding `move` message, that is, the move latency, can result in game clocks not being properly synchronized. To prevent this situation, `move` messages include the thinking-time of the move. Players receiving a `move` message will correct their copy of the sender's clock by subtracting the difference between the time taken according to the sender (without latency), and the time taken from the receiver's perspective (with latency). To limit malicious use, the maximum allowed correction is set to 1 second.

4 Evaluation

In this section we present the results of an emulation of Tribler-G, which show that our protocol functions properly and scales well.

4.1 Experimental Set-up

The experimental results that are presented in this section were acquired using the fourth generation Distributed ASCI Supercomputer or DAS-4 [12]. DAS-4 is a six-cluster distributed system, each of which has one head node, which is used as a file-server, and a number of compute nodes. The DAS-4 clusters communicate with each other through the use of dedicated 10-Gbps light paths. Furthermore, each cluster has a 1-Gbps connection to the Internet through its local university. Within each cluster, the nodes are connected locally through

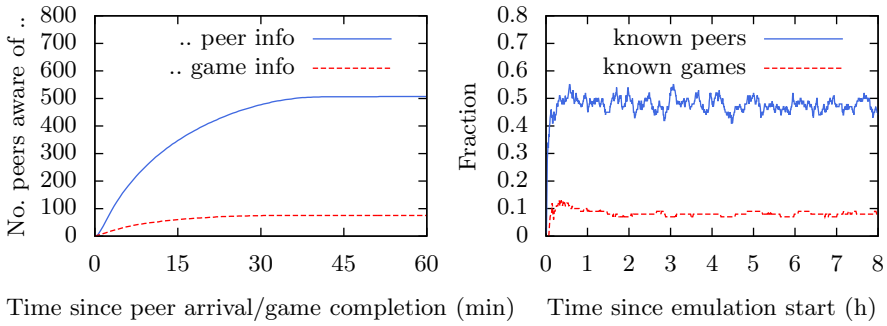

Figure 3. The total number of peers (online and offline) that are aware of a certain peer or of a certain completed game as a function of the time since peer arrival/game completion (left) and the knowledge of peers about games and peers that have been added to the network since they joined the system over time (right).

1 Gbps Ethernet for the normal nodes, and 10 Gbps Ethernet for the head node. DAS-4 runs the CentOS Linux operating system. The nodes of each of the clusters have the following or better configuration: a dual quad-core 2.4 GHz processor, 24 GB memory, 1 TB of local storage, and 18 TB of storage made available by the head node.

4.2 Emulation Environment

We have created the GameTest environment for emulating a network of game-playing peers running the GameCast protocol for the DAS-4 supercomputer. This emulation is made possible by running Tribler-G instances on nodes of the DAS-4. To control what is happening on the emulated network, GameTest requires an input scenario describing when peers join and leave the network. GameTest processes an input scenario a line at a time, with each line specifying the arrival or the departure of a single peer, creating and stopping Tribler-G instances accordingly. Peers arrive in the network with exponentially distributed inter-arrival times and have have uniformly distributed session times. The behaviour of peers during their sessions is determined by the peers themselves, by sending seek/match messages, move/abort/resign messages, and discuss messages, based on random probabilities (for more information see our technical report [13]).

4.3 Experimental Results

In this section we show the results of an experiment performed with GameTest. The arrival rate of peers is 0.27 per second, and the sessions times are uniformly distributed between 10 and 40 minutes. The experiment was performed using 20 DAS-4 nodes and lasted 8 hours, during which 7,618 unique non-recurring peers were part of the network, while the maximum network size was 446 peers. During this experiment, the peers on the network played a total of 7,736 games of chess. These numbers are of the same order of magnitude as those of the FICS. We will show results showing the knowledge of peers about other peers

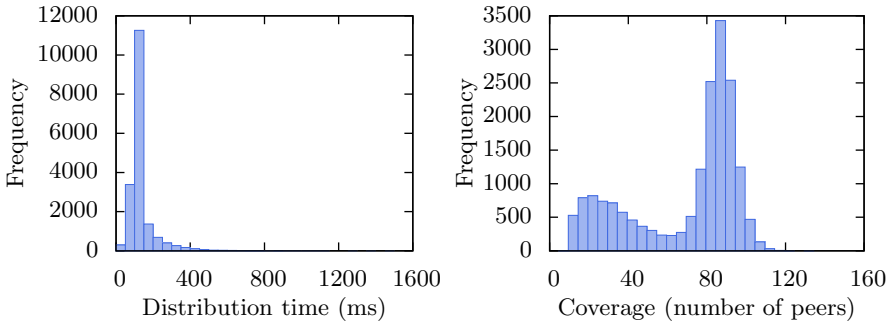

Figure 4. The distribution time and network coverage of random peer invites.

and completed games peers gained through GameCast, the performance of the game agreement process, and the bandwidth used by peers who run GameCast.

To gain insight into the number of peers that are aware of certain information at a particular time, we have monitored the number of peers that are aware of a certain peer or of a certain finished game as a function of time since its insertion into the network. The results, displayed in Figure 3 (left), show that on average, 5 minutes after a peer arrives in the network, well over 100 peers are aware of its existence. It is because of this behaviour that users will be able to invite other players within the network a short time after starting Tribler-G. For games, the behaviour is similar, but less pronounced due to that fact that games are only distributed by their owners (see Section 3.5). Nonetheless, around 10 minutes after a game has finished and starts to be distributed, 50 peers are aware of it. This is more then we initially expected since each peer may only distribute its games to a maximum of 10 connected game buddies and 10 connected random peers. However, since many peers join/leave the network over time, the connected game buddies and random peers also change frequently, leading to a much larger number of peers that are aware of a certain game. Since peers are only online for a relatively small period of time (on average 25 minutes), the number of peers that receive certain information quickly stops increasing. Note that since games are completed after a peer is already online for some time, it will be distributed for a smaller period of time, leading to game information converging much faster.

Figure 3 (right) shows the knowledge of peers about games and peers that have been added to the network since they joined the system. The fractions of knowledge that peers posses are remarkably stable: on average, peers know about 50% of all peers that have connected to the network since they joined the system, and almost 10% of all games that finished since they joined the system.

When peers in the network wish to play a game against each other, they must first agree on the parameters of the game. Inviting a specific peer on the network is achieved by sending messages in a point-to-point fashion. Inviting a random peer, however, is achieved by spreading a message within a distance of two hops. How fast this message is spread and how many peers are reached is very important when finding a suitable opponent. Figure 4 (left) shows the density of the distribution times needed for the random peer invites that were

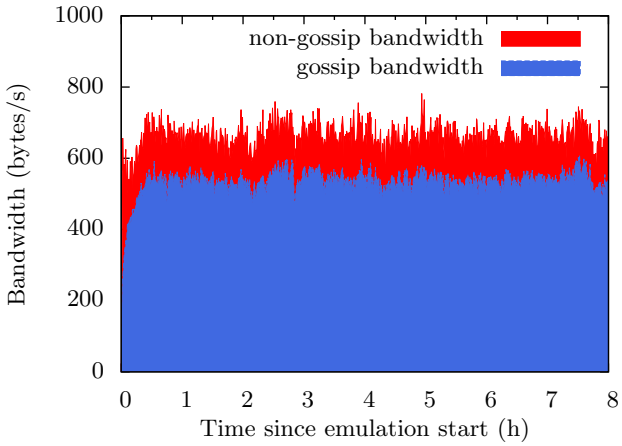

Figure 5. The bandwidth usage of peers on the network over time.

sent during the experiment to reach the complete 2-hop neighborhood of the inviting peer. In almost all cases, the distribution is achieved within just 500 ms. Of course, the DAS-4 has very low network transmission times, so we can expect that real-world distribution times are higher.

Additionally, Figure 4 (right) shows the number of peers that a single random peer invite was able to reach. The minimum number of peers that were reached were about 10 peers. On average, however, the number of peers that were reached was much closer to 90 peers. This is well below the theoretical limit: peers maintain a maximum number of 10 connections with game buddies and with random peers each. Random peer invites are spread over a two hop distance giving us a maximum of 420 peers that can theoretically be reached. However, considering that not all peers will maintain the maximum number of allowed connections, and that many of the neighbors of a peer tend to have connections between each other (i.e., the clustering coefficient is rather high), this number is considerably lower.

In order to test the scalability of the GameCast protocol, we have measured the average bandwidth usage of the peers in the network. We have measured the bandwidth used for information dissemination separately from the other processes. Figure 5 shows the average bandwidth results. We find that the bandwidth for gossip messages is only about 550 bytes/s for the players on average, and that the bandwidth for the remaining messages is well below 200 bytes/s on average. This is in line with our expectations, considering the expected average message length of 5000 bytes for gossip messages and 1000 bytes for the remaining messages, and 0.10 messages/s and 0.15 messages/s for their prospective send rates (see our technical report [13]). Since each peer sends its gossiping messages at a constant rate, an increase in the network size of the peer-to-peer network will not result in an increased bandwidth usage for each individual peer. Furthermore, the bandwidth caused by the execution of non-gossip messages is limited by the number of actions a chess player can make within a certain time period.

We should point out that we have measured the bandwidth at the application level, not taking into account the lower layers of TCP/IP (e.g., TCP headers, IP headers, etc.). Despite the fact that the actual bandwidth usage will be higher, we believe that the GameCast protocol is efficient enough for real world use, where the bandwidth used by GameCast is unlikely to be noticeable by the user.

5 Related Work

Our work is related with an increasing body of research in peer-to-peer gaming [4,8,6]. In contrast to previous work, we focus on simpler in-game worlds and on much richer social interactions. Specifically, we focus on structuring the overlay network such that gamers that frequently play together cluster, thereby implicitly creating virtual communities of gamers, and on supporting social networking activities such as commenting games.

Previous work focuses on complex in-game worlds, which require advanced in-game state management: the world is partitioned among players and special mechanisms ensure the timely delivery of state updates. Closest to our work, PastryMMOG [5] and P2P Second Life [6] use Pastry- and Kademlia-based network structures, respectively; Tribler-G is the first system that provides gaming functionalities over the Tribler network structure [9]. VON [4] partitions the game world using Voronoi diagrams where the centroids are the locations of the players in the game world. Mediator [14] and MOPAR [7] ensure timely delivery of game state updates by dividing the state updates among pre-defined super-peers; super-peers bid for the next work assignment. Donnybrook [8] ensures the timely delivery of state updates under bandwidth constraints by defining areas of interest (AoI); information within the AoI is always transmitted, but information outside the AoI can be summarized or even discarded.

6 Conclusion and Future Work

In this paper we have presented a decentralized system that allows users to play turn-based board games over a peer-to-peer network. Our current implementation, called Tribler-G, is built as an extension to the Tribler file-sharing system, and currently focuses on the game of chess. Tribler-G supports all main gaming features that traditionally only exist in centralized systems, such as the Free Internet Chess Server (FICS), and realizes them in a decentralized setting. The result is a scalable and easy to use application that offers online chess players an attractive alternative to the current centralized services. Experimental results of an emulation show that our approach scales well and functions properly. We have also performed two rounds of user testing, which show that users are overall positive about Tribler-G in terms of software usability (for results see our technical report [13]).

For the future, we would like to improve several technical aspects of Tribler-G, most notably protocol security, rating players and dealing with NAT/firewalls. We also plan to extend Tribler-G's set of features and to increase the number of

games that can by played. And finally, we would like to include a mechanism that allows peers to agree on a random number, which would help generate random content for games that require it (e.g., Scrabble).

References

1. Plunkett, L.: How much has wow cost blizzard since 2004? OnLine (2008) http://kotaku.com/5050300/how-much-has-wow-cost-blizzard-since-2004.
2. Welsh, M., Culler, D.E.: Overload management as a fundamental service design primitive. In: ACM SIGOPS European Workshop. (2002) 63–69
3. Elnikety, S., Nahum, E.M., Tracey, J.M., Zwaenepoel, W.: A method for transparent admission control and request scheduling in e-commerce web sites. In: WWW. (2004) 276–286
4. Hu, S.Y., Chen, J.F., Chen, T.H.: Von: a scalable peer-to-peer network for virtual environments. IEEE Network **20**(4) (2006) 22–31
5. Hampel, T., Bopp, T., Hinn, R.: A peer-to-peer architecture for massive multiplayer online games. In: NetGames, ACM (2006) 48–52
6. Varvello, M., Diot, C., Biersack, E.W.: P2P Second Life: Experimental Validation Using Kad. In: INFOCOM, IEEE (2009) 1161–1169
7. Yu, A.P., Vuong, S.T.: MOPAR : A Mobile Peer-to-Peer Overlay Architecture for Interest Management of Massively Multiplayer Online Games. In: NetGames, ACM (2005) 99–104
8. Bharambe, A., Douceur, J.R., Lorch, J.R., Moscibroda, T., Pang, J., Seshan, S., Zhuang, X.: Donnybrook: enabling large-scale, high-speed, peer-to-peer games. In: SIGCOMM. (2008) 389–400
9. Pouwelse, J.A., Garbacki, P., Wang, J., Bakker, A., Yang, J., Iosup, A., Epema, D.H.J., Reinders, M., van Steen, M.R., Sips, H.J.: Tribler: a social-based peer-to-peer system: Research articles. Concurr. Comput. : Pract. Exper. **20** (2008) 127–138
10. Jelasity, M., Voulgaris, S., Guerraoui, R., Kermarrec, A.M., van Steen, M.: Gossip-based peer sampling. ACM Trans. Comput. Syst. **25**(3) (2007)
11. Glickman, M.E.: Parameter estimation in large dynamic paired comparison experiments. Applied Statistics **48** (1999) 377–394
12. Dutch University Backbone: The distributed ASCI supercomputer 4 (DAS-4) (2011) http://www.cs.vu.nl/das4.
13. Bouman, E.: Tribler-G: A Decentralized Social Network for Playing Chess Online. Master's thesis, Delft University of Technology (2012)
14. Fan, L., Taylor, H., Trinder, P.W.: Mediator: a design framework for p2p mmogs. In: NETGAMES. (2007) 43–48