

Scaling Peer-to-Peer Games in Low-Bandwidth Environments

Jeffrey Pang
Carnegie Mellon University

Frank Uyeda
U. C. San Diego

Jacob R. Lorch
Microsoft Research

Abstract—In peer-to-peer multiplayer games, each peer must send periodic updates of its objects to other peers. Since typical broadband users have little upload bandwidth, updates to each player will be infrequent when there are many players in the game. This leads to choppy and unsatisfying gameplay. Therefore, we propose three techniques to compensate for low upload bandwidth in peer-to-peer games: *focus sets*, *pairwise rapid agreement*, and *guidable AI*. To test these techniques, we implement them and conduct a user study that evaluates the resulting game. We find that our techniques make a game played with low bandwidth significantly more fun than existing techniques, and nearly as much fun as one played on a LAN. Thus, they enable an order of magnitude more players than existing techniques.

I. INTRODUCTION

Online multiplayer games have become an important part of the computing landscape. There is a growing desire to serve these games using the machines of participants themselves rather than with dedicated servers. Using participant machines reduces subscription costs, eliminates dependency on centralized infrastructure, and automatically scales to an arbitrary number of clients. Moreover, one common concern about this approach — the increased risk of cheating — is mitigated by several recently proposed techniques [7].

In proposed peer-to-peer (P2P) architectures [8, 12], each peer is responsible for a subset of all game objects and sends updates about those objects to peers interested in them. Thus, each peer requires enough upload capacity to send updates to all interested peers. However, residential upload rates are typically quite limited, e.g., only hundreds of kb/s in the United States [1]. This is insufficient to support games with dozens or hundreds of interacting players. Area-of-interest filtering can help, but cannot eliminate the problem; e.g., if updates require 16 kbps and a player with 128 kbps is in sight of more than 8 other players, his peer simply does not have enough bandwidth to send updates as often as the game requires. For this reason, we are developing a new architecture, called Donnybrook, to enable large-scale P2P games even in environments with highly constrained bandwidth.

In this paper, we discuss the techniques we use in Donnybrook to improve game playability in low-bandwidth conditions. The main insight that enables our approach is that human attention is bounded by a constant. Thus, the amount of attention in a game grows only linearly with

the number of players, not quadratically. Our techniques enable each peer to send frequent updates only to those players focusing attention on its objects, while preserving timely and consistent interaction and realistic movement for all objects.

To evaluate our techniques, we modify Quake III, a popular first-person shooter (FPS) game, to run on Donnybrook. We present the results of a large user study, which show that Donnybrook substantially increases the enjoyment of P2P Quake III in a low-bandwidth environment. Moreover, our results show that players are nearly as satisfied in a low-bandwidth environment with Donnybrook as they are in a high-bandwidth environment, such as a LAN. Based on this study, we estimate that Donnybrook increases the number of players that can participate in a game by an order of magnitude.

II. BACKGROUND

In P2P architectures for FPS games, like Colyseus [8], each player’s peer is responsible for a subset of all game objects. The peer responsible for an object runs the *primary* of the object, and each other peer interested in it maintains a *replica*. The responsible peer sends updates to replicating peers each *frame*, which typically occurs every 50 ms. An update encodes the object’s changes since the last frame. For example, Bob’s peer may be responsible for Bob’s avatar and missiles that he fires. When Alice can see one of these objects, her peer creates a replica of it. Bob’s peer sends Alice’s peer periodic updates so Alice always sees an up-to-date version.

When the sender has insufficient bandwidth to send updates every frame, it must skip some. Due to the fast-paced nature of FPS games, if the rate drops even slightly below the nominal 20 updates per second, transitions between updates appear “choppy.” For example, because current dead-reckoning [14] techniques cannot handle delays longer than a few hundred milliseconds in FPS games, player avatars will appear to warp between positions instead of running smoothly between them.

To address this problem, existing architectures use area-of-interest filtering to limit the number of updates a peer must send. That is, when two players cannot see each other, their peers need not exchange updates. However, in many game maps, all players are always visible. Even in maps that are partitioned, the popularity of different areas follows a power-law distribution [8], so some areas will have many players visible to each other. Therefore, additional techniques to compensate for low-bandwidth

environments are essential. Donnybrook employs several such techniques.

III. DONNYBROOK

Donnybrook’s design is based on three principles, each of which suggests one of our three techniques:

Players have bounded attention. A human has a fixed “attention budget” and, hence, can only focus on a constant number of objects at the same time [10, 15]. For example, even in a large firefight, a player will tend to focus on his or her current target. This principle implies that the aggregate amount of attention in the game grows only linearly with the number of players, not quadratically. As a consequence, there is always sufficient capacity, within a constant factor, to maintain update rates proportional to player attention. Thus, an object’s update rate to a player should vary based on his or her attention on the object. Our *focus set* technique implements this differentiation in update rates.

Interaction must be timely and consistent. The focus of online games is player interaction. Thus, it is critical that interactions occur in a timely and consistent fashion. For example, when a player kills another, both players should observe the death immediately. Players expect targets of lethal shots to die within milliseconds, so even small delays are jarring. Moreover, if an interaction is not observed consistently, then out-of-band channels such as chat may reveal the inconsistency. This principle suggests prioritizing interactions, i.e., inter-object writes, over other updates. Our *pairwise rapid agreement* technique achieves this prioritization.

Realism should not be sacrificed for accuracy. An out-of-focus object that violates game realism is more likely to be noticed than one that is portrayed with small inconsistencies. For example, it is more important to ensure that the change of a replica’s position obeys game physics than to minimize its error with respect to the primary. Our *guidable AI* technique achieves this realism. Guidable AI makes replicas act in a realistic manner between updates, unlike traditional replicas which only perform dead-reckoning of position between updates.

The following subsections discuss each of these three techniques in detail.

A. Focus Sets

Since player attention is limited, we vary update rates based on estimated player attention. More formally, let A_{ij} be an *attention-value* representing the amount of attention peer i estimates that player i has on player j . Peer i occasionally computes and sends A_{ij} to peer j , piggybacking on updates.

Peer j uses the received attention values to decide how often to send updates to peers. It sends updates to peers with the highest attention-values every frame, and to the rest at whatever frequency is possible given the remaining bandwidth. We call the set of peers receiving per-frame

updates the *focus set*. The frequency of updates to peers not in the focus set is the best-effort rate.

A fixed fraction of bandwidth F_α is assigned to the focus set. During each frame, peer j computes the number of bytes it can send that frame, multiplies this by F_α , then divides by the update size. This gives n , the allowable size of the focus set for that frame. That frame’s focus set contains the n peers with highest A_{ij} values. After sending updates to those peers, peer j uses the remaining bandwidth to send updates to the peers updated least recently.

We compute the attention-value as the weighted sum of metrics that we believe are correlated with player attention. In other words, $A_{ij} = \sum_{k=1}^m w_k F_{ij}^{(k)}$, where $F^{(1)}, F^{(2)}, \dots, F^{(m)}$ are our m metrics, and w_k is the weight for metric $F^{(k)}$. We currently use three metrics:

Proximity. Players are more likely to pay attention to close objects. Thus, we use the following proximity metric:

$$F_{ij}^{(1)} = \max\{(1 - \text{dist}(i, j)/D_{max})^{1.5}, 0\},$$

where D_{max} is about 1/2 the diameter of a map. The exponent 1.5 is based on the rate at which objects become visually “smaller” as distance increases.

Aim. Players are more likely to pay attention to objects they are aiming at. Let the *aim deviation* a_{ij} from player i to player j be the angle between player i ’s forward vector and the vector from player i to player j . Since instantaneous aim can be erratic, our aim metric uses the more stable \hat{a}_{ij} , an exponentially weighted moving average of a_{ij} . Our aim metric is

$$F_{ij}^{(2)} = \max\{(1 - \hat{a}_{ij}/45^\circ)^{1.5 \cdot \log(\text{dist}(i, j))}, 0\}.$$

We assume players are paying little attention to objects beyond the 90° visible cone. We multiply the fall-off rate by $\log(\text{dist}(i, j))$ because more objects are visible at larger distances and they appear smaller; thus, a player must aim more carefully to discriminate between them.

Interaction Recency. Since interaction is the most important aspect of multiplayer games, players who recently interacted should be paying attention to each other. Our metric computes

$$F_{ij}^{(3)} = \begin{cases} e^{-t_{ij}/1 \text{ sec}} & \text{if } t_{ij} \leq 3 \text{ sec} \\ 0 & \text{otherwise} \end{cases}$$

where t_{ij} is the time since an interaction between players i and j as defined in §III-B. FPS games are fast-paced so we bound the influence of interactions to a few seconds and make it fall off quickly.

Different metrics are more important in different situations, so we vary the weights $\{w_k\}$ based on player state. Normally, we weight interaction recency about 1.5 times more than proximity and aim, but we raise the weight of proximity when wielding a melee weapon and raise the weight of aim when wielding a sniper weapon. Our

weights and metrics yield good playability despite limited tuning effort on our part.

B. Pairwise Rapid Agreement

When one object W modifies another object T , we call this an *interaction*, with W the *writer* and T the *target*. For example, if one player shoots another, the shooter acts as a writer because he decrements the target’s health field. Our policy is that upon an interaction, the writer’s peer immediately communicates with the target so they can quickly agree on the state of the changed object. We call this *pairwise rapid agreement* (PRA).

A PRA consists of an asynchronous RPC from the writer’s peer to the target’s peer. The target’s peer applies the write to its primary copy and replies with the new target state. Sometimes the resulting state change is predictable, so the reply can be only a few bits or even completely elided. Note that the writer’s replica of the target may be inconsistent, and the write may turn out to be impermissible. For instance, a player may try to pick up an important item that has already been taken by another player. In this case, the target simply responds that the write did not happen.

PRA’s are feasible because they do not appreciably diminish the bandwidth budget for updates. Interactions occur at human timescales, so they are infrequent compared to updates. Also, PRA’s involve only one-to-one, not broadcast, communication.

We found only four interaction types in Quake III: one player damages another, one player dies and increments another’s score, a player picks up an item, and a player opens a door.

To enable PRA’s for weapon damage, we assume that a peer knows who gets hit by weapons fired by the local player. In other words, the shooter decides whom his fire hits, based on his or her possibly inaccurate view of the target’s location. Basing this decision on the shooter’s worldview is common in FPS games, since players with high-latency network connections find it unrealistic to fire directly at a player and miss [6].

C. Guidable AI

A player replica whose peer is not in the focus set of its primary receives infrequent updates. As noted earlier, simply using dead-reckoning between infrequent state snapshots would make replicas appear “choppy” and unrealistic. Therefore, instead of sending state snapshots, the primary sends such replicas *guidance* about how the replica should act until the next update. The replica, in turn, uses a *guidable AI* to simulate the primary. This AI ensures that movements appear realistic, while attempting to coarsely approximate the primary’s state via the guidance. The guidance a primary sends to replicas consists of a *prediction* and a set of *action counters*, which we now describe.

A prediction is the primary’s estimate of its state t seconds in the future, where t is the expected time

between consecutive best-effort updates to the same peer. In Donnybrook, we predict two fields: position and facing direction. To predict position, we simulate where the player will be in t seconds if his or her input does not change; we found this to be sufficiently accurate when $t \leq 1$ sec. Facing direction is expressed as the angle between a player’s actual aim and his or her estimated target. The target is estimated as the other player who minimizes \hat{a}_{ij} as defined in §III-A. We predict facing direction relative to another player rather than as an absolute direction so that despite inconsistencies in the locations of replicas, the directions they face relative to other replicas will look fairly correct.

Actions the replica should replay, like firing shots, death, and respawning, are not continuous like the previous two properties. To accommodate these, guidance includes counters of how many times each action occurred so that the replica can replay them the correct number of times. However, note that a replica is not permitted to modify other objects. So, for instance, if it replays shots fired, these shots must be “blanks.”

Most modern FPS games contain AI routines for computer-controlled players known as bots. Since bot AI code attempts to emulate real player behavior, it is well suited for our purpose of making replicas act realistically. Thus, to implement guidable AI we do not have to write new AI routines; we can use ones already present in the game. For instance, a peer receiving guidance can use the bots’ AI path-finding routine to have its replica move to the predicted position and turn to face the predicted target with the estimated deviation. Additionally, it can use other bot AI routines to perform actions required by action counters.

Although guidable AI makes replicas move realistically, it can lead to high inaccuracy. Thus, it is only used for players not in focus. When a player focuses on another player, its peer begins receiving frequent updates from that player’s primary, causing the replica of that player to converge to the correct state, as follows.

When a replica begins receiving state snapshots from a primary, applying them directly might require unrealistic warping from its inaccurate position to the correct position. Thus, the primary continues to send guidance along with the state snapshots. Since guidance arrives frequently, the replica will soon converge to the correct state. We consider it converged when it is within a player object diameter of the correct position; other properties do not matter because they can be interpolated quickly without jarring effects. When the replica has converged, it tells the primary to stop sending guidance, and it starts applying state snapshots directly.

IV. EVALUATION

The ultimate goal of games is user satisfaction. If a game can support many players with little upload bandwidth but is not fun, then the capability is meaningless.

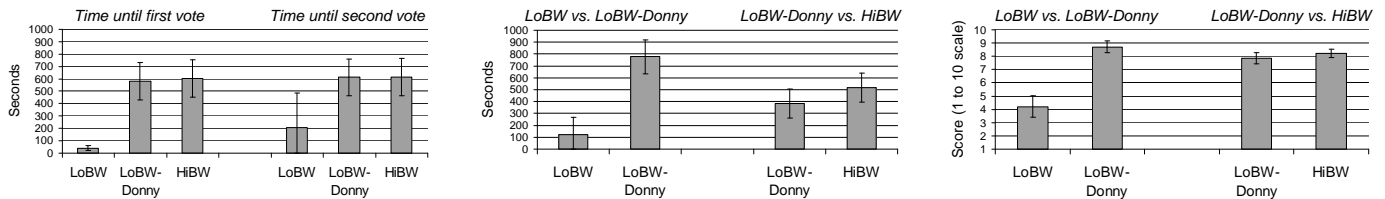


Fig. 1. User satisfaction experiment results. From left to right, these show (a) time until the votes to switch, (b) total time spent on each version, and (c) self-reported scores. Error bars show 95% confidence intervals.

Therefore, we evaluate Donnybrook’s support for such conditions based on how fun it is under those conditions.

Our evaluation of Donnybrook answers two main questions. First, how much more enjoyable is Quake III in a low-bandwidth setting with Donnybrook than with current techniques? Second, is a game with Donnybrook in a low-bandwidth setting comparably enjoyable to a game in an ideal setting, like a LAN? To answer the first question, we compare LoBW-Donny, a low-bandwidth setting using Donnybrook, to LoBW, a low-bandwidth setting using current techniques. To answer the second, we compare LoBW-Donny to HiBW, a high-bandwidth setting.

We perform these evaluations using a modified version of Quake III, a popular FPS game. This modified version can be configured to use either Donnybrook or the current state-of-the-art, i.e., dead reckoning. Quake III is not normally a P2P game, so our modified version is a simulation of a P2P game. Each peer is emulated with a virtual Quake III server with its own copy of the game state. The virtual servers run in a single process, which emulates a network between the peers.

A. User Satisfaction

First, we study overall user satisfaction.

Methodology. A real-life scenario in which players express a preference between versions of the same game is when they select a public server to play on. Players typically choose from a list of servers; if they become dissatisfied with a server, they leave and try another. Our user study emulates this scenario, as follows.

Pairs of volunteers sign up to play a game of Quake III. They are told that there are two Quake III servers with different network characteristics they can choose from. In each experiment, these servers either use LoBW-Donny and LoBW, or LoBW-Donny and HiBW, but we do not tell them how the servers differ. They begin playing on one server and can vote to switch. They switch versions when both players vote to do so, and they switch back and forth as often as they wish. Player scores are transferred from one version to the next, so there is no incentive for the losing player to switch. Also, to avoid bias from the order that versions are played, for each of the two experiment types half of the pairs begin on one version and half begin on the other. After playing 15 minutes total on both versions, they play a new 5-minute game on

the least-used server so they have enough experience to compare the two.

To simulate a large game, we use 30 bots in addition to the two human players. Each bot or player is treated as a separate peer in the virtual network, but all bots execute on a single virtual server due to limited computational resources. This means bots never see inconsistencies, but we do not believe that this substantially affects our results since it does not change the update volume that is sent between peers and we only measure the experience of humans.

We take several steps to encourage human interaction. We make human avatars easily distinguishable from bots. We make it practically impossible to win by killing only bots by making killing a human worth ten times more points. We give the winner a token prize. Finally, we encourage players to communicate verbally as if voice chat were enabled.

In LoBW and LoBW-Donny, each peer has 108 kbps of outbound bandwidth. We choose this capacity because it enables a focus set size of four and a best-effort rate of one update per second, settings that we found satisfactory during our own play-testing. HiBW has no bandwidth limit. In all scenarios, the RTT between each pair of peers is 20 ms. All games use the popular map `q3dm17`, in which all players can see one another.

We conduct 12 trials of LoBW vs. LoBW-Donny and 32 trials of LoBW-Donny vs. HiBW, using a total of 88 different participants. In general, participants are very familiar with FPS games, with 62% reporting that they played FPS games every day at some point in their life and 87% reporting at least once per week. Nonetheless, we give all players 8 minutes to practice playing Quake III in the HiBW scenario before starting our experiment.

Results. We use four metrics, described below, to measure user satisfaction. Results given by all four metrics support two main conclusions. First, players have a much more favorable impression of low-bandwidth games with Donnybrook than without. Second, with Donnybrook, players are about as satisfied with a low-bandwidth game as they are with an unlimited-bandwidth game.

Time until switch. Players are likely to leave an unsatisfactory version sooner than a satisfactory one. Fig. 1(a) shows the time until the first player in each trial votes to switch from the initial version and the time until both

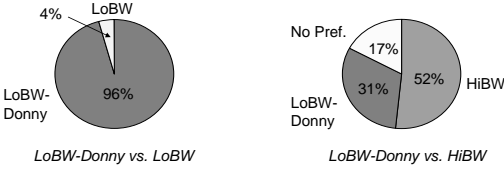


Fig. 2. Self-reported version preference.

players vote to switch, on average. Clearly, players are dissatisfied with LoBW very quickly — on average, the first vote comes at 40 seconds. In contrast, players vote to leave LoBW-Donny after about the same time as they would have left HiBW, on average.

Total time spent. Players are likely to play longer in a game that they find more enjoyable. Fig. 1(b) shows the average amount of time spent on each version in the two different experiment types. Given the choice between LoBW and LoBW-Donny, players overwhelmingly choose to spend time on LoBW-Donny. When choosing between LoBW-Donny and HiBW, players spend slightly more time on HiBW, though this difference is statistically insignificant given our small sample size.

Self-reported score. After the experiment, players rate their enjoyment of the two versions on a 1–10 scale. Fig. 1(c) shows the average scores given to each version in the two different comparisons. On average, LoBW-Donny is preferred by 4.5 points over LoBW, whereas the difference LoBW-Donny and HiBW is statistically insignificant.

Self-reported preference. Finally, each player is asked which version they like better. Fig. 2 shows user preferences in the two different comparisons. Again, these results corroborate our two conclusions. One anomaly is the single player who prefers LoBW to LoBW-Donny. This is because, in his words, “It brings back my memory of playing Quake I over a 28.8k modem.”

B. Fairness

In this section, we evaluate an important aspect of user satisfaction, namely, how Donnybrook affects *fairness*, i.e., how well game outcome reflects player skill. Low update rates can lead to unfairness because inconsistency in a player’s replica can prevent it from accurately reflecting the player’s actual actions.

Methodology. The relative scores of different players usually reflect their relative skills. Thus, to evaluate Donnybrook’s effect on fairness, we study how well it preserves relative scores. Since it would take too long to obtain sufficient gameplay samples from humans, we use Quake III bots. Quake III provides 32 different bot models, each available at five skill levels. Each model has different characteristics, such as weapon preference. Different skill levels of a model have different refinements of these characteristics; e.g., higher skill levels often correspond to better aim.

In our experiment, bots with random models and skill levels play a game for 24 hours using HiBW. The scores at the end of this game are the *expected* scores. Then,

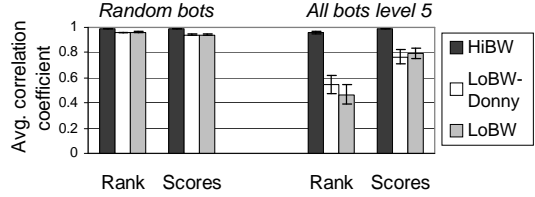


Fig. 3. Average hourly correlation coefficient between expected scores and actual scores under various conditions. Error bars show 95% confidence intervals.

the same bots play one game using LoBW-Donny and one using LoBW. For each hour of these games, we compute the correlation of the scores and the expected scores. The average hourly correlation in each game is our fairness metric. To illustrate how high a measure is reasonable to expect, we report the same metric for the HiBW run. The difference between this value and 1.0 shows the deviation that random effects can produce.

We compute two correlation measures: Spearman’s rank correlation coefficient and Pearson’s correlation coefficient. The former considers relative rankings, while the latter considers absolute score values. In both, a value of -1 indicates perfect negative correlation (worst case), 0 no correlation, and +1 perfect positive correlation (best case).

In this experiment, unlike that of §IV-A, we are concerned with bot interaction. Therefore, we run each bot in a separate virtual server so they can see inconsistencies. Limited computational resources restricts this setup to 16 bots. Our user satisfaction study indicates that players enjoy Donnybrook with a focus set size of four and best-effort update rate of 1 per second, so we set upload bandwidth and F_α accordingly in LoBW-Donny and use the same upload bandwidth in LoBW. We also use the same q3dm17 map.

Results. The *Random bots* portion of Fig. 3 shows the results of this experiment. We see that the LoBW-Donny and LoBW correlation coefficients are very high, with 0.96 for ranks and 0.94 for scores. Thus, it seems fairness is not substantially reduced by low update rates, whether or not Donnybrook is used.

However, upon further investigation, we discover that the reason for the high fairness is that scores tend to be ordered by skill level in all scenarios. Thus, we perform the experiment again using bots with random models but all at skill level 5. The *All bots level 5* part of Fig. 3 shows the coefficients in this experiment, which are much lower for LoBW and LoBW-Donny, but still positive. In addition, the difference between LoBW and LoBW-Donny coefficients are statistically insignificant.

We conclude that Donnybrook preserves fairness at a coarse level, but may cause unfairness in matches between players with similar skill levels. Donnybrook has similar fairness to existing architectures in low-bandwidth conditions, so guidable AI does not seem to cause a reduction in fairness. We discuss avenues we are pursuing to improve Donnybrook’s fairness in §VI.

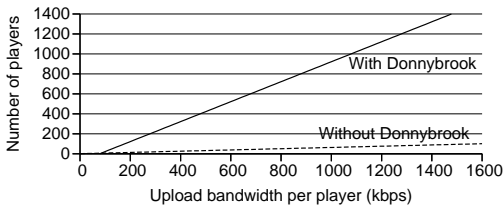


Fig. 4. Projected scalability of Donnybrook

C. Discussion

Our user satisfaction experiment suggests that with Donnybrook, a game is enjoyable if there is sufficient bandwidth for a focus set size of four and a best-effort rate of one update per second. On the other hand, without Donnybrook, each peer needs enough bandwidth to broadcast updates every frame. Fig. 4 shows a projection of the number of players supported as a function of available upload bandwidth per peer. It shows that Donnybrook can enable an order of magnitude more players in a game.

The projection assumes all players can see each other. If not, and peers use distributed area-of-interest filtering [8] to only send updates to peers who can see them, then the number of players is the number in an area of interest.

V. RELATED WORK

The classic solutions for limited communication bandwidth in games are dead reckoning and interest management. There has been much research both in evaluating dead reckoning schemes [13, 14, 17] and in improving them with techniques such as timestamping [4], error equalization [5], or use of position history instead of instantaneous velocity [16].

Interest management means sending updates only to those players who have interest in the object in question. Prior research in interest management includes separation of relevance based on game state from relevance based on network conditions [2], multi-tier interest management with successively increased resolution [3], and publish-subscribe architectures for area-based interests [8, 11]. Our focus set approach is a refinement of interest management, where update frequency is varied based on attention.

A consequence of guidable AI is that different players see different views of the world. This is also the case in Rendezvous [9], a highly optimistic, P2P consistency mechanism for high-latency environments. Unlike Rendezvous, guidable AI causes objects that are focused on to converge to their true states.

VI. SUMMARY AND FUTURE WORK

This paper presents techniques to compensate for limited upload bandwidth in large-scale peer-to-peer games. Our results demonstrate that users prefer Donnybrook to existing techniques for compensating for low bandwidth, and that Donnybrook makes low-bandwidth games nearly as much fun as games with unconstrained bandwidth. Consequently, Donnybrook enables far more players in a peer-to-peer game.

One avenue of future work is to improve Donnybrook’s fairness. We may be able to leverage machine learning techniques to improve each guidable AI’s representation of a player. In addition, we can refine PRA; e.g., we have a scalable technique that lets the target of a missile, rather than the shooter, decide whether a hit occurs, so that missile dodging skills are preserved.

Another avenue of future work is to handle skewed distributions of player attention. Fixed-size focus sets do not work well when many players are paying attention to a single player, as can happen in a capture-the-flag game. Fortunately, since the aggregate amount of attention in the game is constant, in this situation there must be some peers with little attention on them and, thus, with spare upload capacity. To address this problem, we are working on an overlay multicast scheme to let peers share their upload capacity in a latency-sensitive manner.

VII. ACKNOWLEDGEMENTS

We are deeply indebted to John Douceur for his excellent ideas and support. We also thank Bryan Parno for helping us “test” Donnybrook, the players in our experiments for participating in our study, and the anonymous reviewers for their helpful suggestions.

REFERENCES

- [1] Survey of <http://www.broadbandreports.com>, Oct. 2006.
- [2] AARHUS, L. ET AL. Generalized two-tier relevance filtering of computer game update events. In *NetGames* (2002).
- [3] ABRAMS, H. ET AL. Three-tiered interest management for large-scale virtual environments. In *Virtual Reality Software and Technology (VRST)* (1998).
- [4] AGGARWAL, S. ET AL. Accuracy in dead-reckoning based distributed multi-player games. In *NetGames* (2004).
- [5] AGGARWAL, S. ET AL. Fairness in dead-reckoning based distributed multi-player games. In *NetGames* (2005).
- [6] BERNIER, Y. W. Latency compensating methods in client/server in-game protocol design and optimization. In *Game Developers Conference* (2001).
- [7] BHARAMBE, A. Scalable and Secure Architectures for Online Multiplayer Games. Thesis Proposal, Apr. 2006.
- [8] BHARAMBE, A. ET AL. Colyseus: A distributed architecture for online multiplayer games. In *NSDI* (May 2006).
- [9] CHANDLER, A. ET AL. On the effects of loose causal consistency in mobile multiplayer games. In *NetGames* (2005).
- [10] COWAN, N. The magical number 4 in short-term memory: A reconsideration of mental storage capacity. *Behavioral and Brain Sciences* 24, 1 (2001).
- [11] FIEDLER, S. ET AL. A communication architecture for massive multiplayer games. In *NetGames* (2002).
- [12] KNUTSSON, B. ET AL. Peer-to-peer support for massively multiplayer games. In *INFOCOM* (July 2004).
- [13] PALANT, W. ET AL. Evaluating dead reckoning variations with a multi-player game simulator. In *NOSSDAV* (2006).
- [14] PANTEL, L. ET AL. On the suitability of dead reckoning schemes for games. In *NetGames* (2002).
- [15] ROBSON, J.G. ET AL. Probability summation and regional variation in contrast sensitivity across the visual field. *Vision Research* 21, 3 (1981).
- [16] SINGHAL, S. K. ET AL. Exploiting position history for efficient remote rendering in networked virtual reality. *Presence* 4, 2 (1995).
- [17] YASUI, T. ET AL. Influence of network latency and packet loss on consistency in networked racing games. In *NetGames* (2005).