Donnybrook: Enabling Large-Scale, High-Speed, Peer-to-Peer Games

Ashwin Bharambe*, John R. Douceur†, Jacob R. Lorch†, Thomas Moscibroda†,
Jeffrey Pang*, Srinivasan Seshan*, and Xinyu Zhuang*
*Carnegie Mellon University, Pittsburgh, PA †Microsoft Research, Redmond, WA
{ashu,jeffpang,srini,xinyuz}@cs.cmu.edu {johndo,lorch,moscitho}@microsoft.com

ABSTRACT

Without well-provisioned dedicated servers, modern fast-paced action games limit the number of players who can interact simultaneously to 16–32. This is because interacting players must frequently exchange state updates, and high player counts would exceed the bandwidth available to participating machines. In this paper, we describe Donnybrook, a system that enables epic-scale battles without dedicated server resources, even in a fast-paced game with tight latency bounds. It achieves this scalability through two novel components. First, it reduces bandwidth demand by estimating what players are paying attention to, thereby enabling it to reduce the frequency of sending less important state updates. Second, it overcomes resource and interest heterogeneity by disseminating updates via a multicast system designed for the special requirements of games: that they have multiple sources, are latency-sensitive, and have frequent group membership changes. We present user study results using a prototype implementation based on Quake III that show our approach provides a desirable user experience. We also present simulation results that demonstrate Donnybrook's efficacy in enabling battles of up to 900 players.

Categories and Subject Descriptors

C.2.4 [**Computer-Communication Networks**]: Distributed Systems

General Terms

Algorithms, Experimentation, Human Factors

Keywords

computer games, doppelgängers, interest sets

1. INTRODUCTION

Computer games are a large and growing business, with revenues almost twice that of movies [41]. The current evolution in networked games is marked by three significant trends: *large scale*, as in the massively-multiplayer World of Warcraft; *high speed*, as in the fast-paced Halo 3; and *peer management*, as in Xbox Live's use of player machines instead of dedicated servers to reduce the cost of hosting games [26]. However, there are significant technical barriers to achieving all three properties at once, and no existing game does so. In this paper, we describe *Donnybrook*, our system that enables such games, with hundreds of players interacting in one area at the same time. Donnybrook contributes solutions to three important problems in peer-managed environments: insufficient capacity, resource heterogeneity, and interest heterogeneity.

Insufficient capacity. Broadband connections are generally asymmetric, so the key limitation is upload capacity. As game scale increases, the aggregate upload capacity increases linearly, but communication demands grow *quadratically*. Each player must receive updates about every other player whenever state changes, e.g., at $\sim 12n$ kb/s per player for n players in Quake III. Thus, even p2p architectures [4, 22], where each peer only sends updates for its own player, would need 10 Mb/s per peer in a 900-player game.

A conventional approach to reducing bandwidth demand is *area-of-interest (AOI) filtering*, meaning each player sends updates only to nearby players [4, 16]. This approach works well when the number of players in any given area is strictly limited. However, no such limit occurs naturally, because population density in real games follows a power law [35]. Thus, game designers must artificially discourage player clustering, thereby precluding certain classes of interesting game play, such as epic battles [6, 40].

In Donnybrook, we introduce the notion of a player's *interest set*, the set of other players to whom he is paying attention. Unlike an area of interest, which contains more players as player density increases, an interest set always remains small due to the limits of human attention. In Donnybrook, a player receives updates only from the players in his interest set, thereby enabling epic battles with hundreds of players in close proximity. To deal with visible players outside the interest set, Donnybrook represents them with low-behavioral-fidelity *doppelgängers*, which require only low-rate *guidance* information rather than continuous updates.

Resource heterogeneity. Each player must still send guidance roughly once per second to each other player, to prevent views from diverging. Donnybrook disseminates guidance using multicast trees so that machines with spare capacity can forward guidance for machines with insufficient capacity. Forwarding guidance without considering total delay, however, would not ensure players receive fresh guidance before older guidance expires. The key challenge we address is how to schedule guidance dissemination in a way that ensures timely delivery, minimizes bandwidth use, and is resilient to churn.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
SIGCOMM'08, August 17–22, 2008, Seattle, Washington, USA.
Copyright 2008 ACM 978-1-60558-175-0/08/08 ...$5.00.

Interest heterogeneity. If one player is interesting to many others, he may not have sufficient capacity to send updates to them all. A player can become highly interesting in many ways, e.g., by holding a flag in a capture-the-flag game or by being in a prominent location. Fortunately, the aggregate capacity required to send updates is always proportional to the number of machines, since each player is always interested in a constant number of others. Thus, we again can leverage spare upload capacity to help forward these updates. However, in contrast to the guidance distribution problem where receiver sets change rarely, a player's interest set changes often, with average turnover of 68% per second according to our user study. The key challenge we address is how to deliver updates in the face of this substantial interest flux, yet still maintain 100–150 ms latency and be resilient to churn.

This paper describes the novel techniques Donnybrook uses to address each of these problems, providing a complete solution that enables peer-managed, large-scale, high-speed games. We present user study results that show that our approach provides a desirable user experience, and simulation results that demonstrate its efficacy in enabling battles of up to 900 players.

The rest of this paper is organized as follows. §2 provides background. §§3–5 describe our techniques for addressing the three challenges listed above. §6 presents a user study that validates our use of interest sets, and §7 presents a simulation evaluation of the Donnybrook system. §8 describes prior work, and §9 concludes and discusses future work.

2. BACKGROUND

2.1 First-person shooter (FPS) games

Our goal is to scale high-speed games involving fast-paced interaction. We limit our discussion to the first-person shooter genre since most fast-paced games fall into this category. In an FPS game, each player controls an avatar who navigates through an area, picks up objects such as weapons, and shoots and is shot by enemy avatars. Each player's view is rendered using *updates* he receives about the other objects in the game.

Update messages. An object's state consists of many fields, such as position and current weapon. Every object changes once per 50-ms game *frame*. Since most fields change rarely, updates are compressed via *delta-encoding*, i.e., sending only fields that have changed since an earlier state the receiver is known to have. Typically, this means sending fields changed since the previous state, but it may also include fields changed since the state two frames earlier to cope with the loss of one update. If two or more update packets are lost in a row, then a non-delta encoded update is required to come up to date. Usually, only player controlled objects such as avatars change frequently, so updates about these objects dominate. In Quake III, the mean update size is 32 bytes with regular delta-encoding, 36 bytes with 2-frame delta-encoding, and 196 bytes without delta-encoding. Packet headers add additional overhead, e.g., ∼44 bytes in Xbox Live [29]. This is more than UDP/IP header sizes because of packet authentication for cheat prevention.

Update frequency. Players in FPS games move rapidly and must react quickly to their surroundings. As a consequence, limiting *lag*, the time difference between a player's visible state and his actual state, is crucial for a satisfactory experience. Studies have shown that lag more than ∼100 ms decreases user satisfaction and degrades player performance, and lag of 200 ms is unacceptable [3]. Thus, a player receives state updates about all objects they are interested in every frame over UDP. In addition, many game matchmaking services group players with others in the same geographic region [5, 15] to reduce latency.

2.2 Peer-managed games

Peer management runs games using the resources of players' machines. Dedicated servers still provide game-external support such as authentication and matchmaking, but these services use few computational and bandwidth resources. Peer management reduces play cost, allows small publishers to enter the market, and reduces the investment risk in games of unknown popularity.

Due to the high frequency of FPS game updates, the key limitation to scale in peer-managed games is the players' upload capacity. A player with broadband will typically have upload capacity equal to that of his access link (often < 1 Mb/s), since this capacity is typically an order of magnitude smaller than Internet bottlenecks [1]. Even players with more capacity, e.g., those playing from universities, will likely restrict their aggregate upload rate to a few Mb/s to avoid antagonizing network administrators [28, 33]. We call a peer's upload limit its *rate limit*.

Modern peer-managed FPS games typically have one peer send updates about all game objects, including player avatars, to all other peers. Due to the lack of IP multicast support between most Internet hosts, practical deployments must send these updates in a point-to-point manner. Thus, this bandwidth requirement scales quadratically with the number of players. As a consequence, these games can not allow more than ∼16–32 players at the same time.

Instead of having a single peer send updates about all objects, each peer may manage a subset of objects, such as its player avatar [4, 22]. In such *p2p* architectures, a peer holds the authoritative copies of objects it manages, and non-authoritative replicas of objects managed by other peers. It sends updates of only the objects it manages to other peers, so they can update their replicas. Donnybrook uses such a p2p architecture so it can leverage the resources of all peers. Unfortunately, these resources only grow linearly with the number of peers while bandwidth demand grows quadratically, so we need to do more to achieve the scale we desire.

3. INTEREST SETS

In this section, we describe how we leverage limits of human cognition to reduce bandwidth requirements. Specifically, using our techniques, bandwidth required to send updates scales *linearly* with the number of players in the battle. Quadratic scaling is only required for messages sent at substantially lower frequency, thereby enabling epic-scale battles.

3.1 Overview

A human can only focus on a constant number of objects at once [11, 37]. Consequently, a player is likely interested in a few others and notices only rudimentary information about everyone else. Thus, he only needs high-fidelity replicas of those he is interested in.

We leverage this observation by having each player send updates only to those who are interested in him. To de-

termine who these players are, we estimate each player's *interest set*, the set of players he is most interested in. The aforementioned observation suggests this set is of constant size; based on the results of a user study, we use a set of five (see §6). Those who are not interested in a player receive information at a much lower frequency, roughly once per second, which we make possible by using a low-fidelity replica called a *doppelgänger*. We discuss details of these techniques below.

3.2 Estimating interest sets

Each player's game client estimates how much attention he is paying to each other player. We define his interest set to be the five players he is paying most attention to. Specifically, let A_{ij} be an *attention value*, the amount of attention player i's game client estimates that player i is paying to player j. Then, player i's interest set is the set of players with the five highest A_{ij} values.

Player i recomputes his interest set every frame. When a new player j enters player i's interest set, i sends j a `subscribe` message to indicate j should send i frequent updates. An `unsubscribe` message is sent when j leaves the set. We find that an interest set is generally stable over several frames, making it a useful predictor of short term interest.

Intuition suggests that the set of players a person is paying attention to has spatial and temporal locality. As a consequence, we compute A_{ij} as the weighted sum of metrics that measure how close i and j are in space and time. In other words, $A_{ij} = \sum_{k=1}^{3} w_k I_{ij}^{(k)}$, where $I_{ij}^{(1)}, I_{ij}^{(2)}, I_{ij}^{(3)}$ are our three metrics, and w_k is the weight for metric $I_{ij}^{(k)}$. The three metrics we use are:

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

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

where D_{max} is the distance at which objects can not be discerned. This metric is based on the assumption that a player's attention on an object is roughly proportional to the visible size of that object, relative to others. In a 2D game world (i.e., a plane), the number of objects potentially visible at distance *dist* is inversely proportional to *dist* since objects can only take up space along the horizon. In a 3D game world (i.e., free space), the number of objects potentially visible is inversely proportional to $dist^2$ since objects can take up space horizontally and vertically. Since FPS maps are somewhere in between a flat plane and free space (e.g., a building with several floors), we choose a fall-off exponent between 1 and 2.

Aim. Proximity is not the only measure of spatial locality because players also have an orientation. They 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 i's forward vector and the vector from i to 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:

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

This metric is based on the assumption that a players' attention is highest on objects in the middle of the screen and falls off for objects are closer to the edges where it becomes 0 (the visible cone is $\sim 90°$). Since objects are three dimensional, the fall-off rate $1.5 \cdot \log(\text{dist}(i,j))$ accounts for the amount aim can deviate and still "point at" an object. Distant objects appear smaller; thus, a player must aim more carefully to reliably hit them.

Interaction Recency. In addition to spatial locality, there is also temporal locality in player interaction. Players who recently interacted are more likely to pay attention to each other. Our metric is:

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

where t_{ij} is the time since the last interaction between players i and j. We define an interaction as any instance where one modifies the state of the other, such as when a shot causes damage. In contrast to the spatial metrics, our play testing suggests that temporal influences on attention fall off much quicker due to the fast-paced nature of FPS games. Thus, this value falls of exponentially rather than polynomially and we bound the influence of interactions to a few seconds.

Weight tuning. Since each $I_{ij}^{(k)}$ is based on a different unit, we use linear weights $\{w_k\}$ to normalize their influence based on play testing. With only several hours of tuning on our part, we found that weighting interaction recency 1.5 times more than proximity and aim yielded good playability independent of game scale in Quake III (see §6). Different metrics are more important in different situations, so we also vary the weights based on player state. For example, a player wielding a melee weapon will be focused on hitting nearby players, whereas a player wielding a sniper weapon will be more focused on shooting players at a distance. Thus, we give the former player a higher w_1; the latter a higher w_2.

We believe these three metrics are sufficient for most FPS games since player interactions are similar. Minor variants, such as an aim metric that takes into account weapons that fire in non-straight-line trajectories (e.g., grenades), should be simple to devise based on game physics and the same locality metrics. Although a different tuning of weights via play testing will likely be required for each game, such parameter tweaking is already a large component of game development, e.g., for balance testing. Other games might also vary weights based on a player's team in cooperative games, possession of important items, and special powers such as heightened senses or flight.

3.3 Doppelgängers

A player receives updates every frame from players in his interest set, but receives information infrequently from everyone else. Since he is not paying attention to these players, they can be rendered at lower fidelity. However, state-of-the-art techniques for extrapolating from missing updates, e.g., dead reckoning [32], cannot handle delays longer than a few hundred milliseconds in FPS games, because player avatars will appear to warp between positions instead of running smoothly between them. Since these avatars can still appear in a player's peripheral vision, if they act in a manner that violates game physics (e.g., sudden, jerky movement), they are likely to draw attention when they should not.

Therefore, to extrapolate the behavior of a remote player that sends only infrequent information, we use a special replica called a *doppelgänger*. A doppelgänger is a *bot*, i.e., a computer-controlled player running on the local machine,

whose goal is to act in a manner that realistically approximates the behavior of the remote player. To implement a doppelgänger, we leverage a game's existing AI routines, which are designed to make bots act realistically. However, unlike standard bots, which are free to act in any way, doppelgängers must attempt to roughly emulate the behavior of a particular remote human. For this purpose, a doppelgänger uses *guided* AI, AI that takes input from the human player it represents.

To enable guided AI, every player sends *guidance*, not updates, to players not interested in him. Guidance is a compact summary of his predicted behavior for the period between now and the next anticipated message, such as where he expects to go, whom he is targeting, and how often he fires his weapon. A doppelgänger uses this information to tailor its behavior. E.g., instead of running in a straight line with the same velocity as before, it uses AI path-finding code to navigate a realistic path to the predicted location. Since guidance is received once per second, a doppelgänger's state is unlikely to deviate substantially from the actual player's state. Thus, unless a player deliberately focuses on the doppelgänger, he is unlikely to notice the difference.

Since the focus of this work is on network support, we omit further details about guided AI that were presented in our workshop paper [31].

3.4 Message dissemination overview

Using interest sets, each player receives frequent updates, at a rate of 20 per second, from only five other players independent of game size. Guidance is sent to all players, but only once per second. Thus, while the the mean bandwidth requirement per player still scales quadratically, it is reduced by a factor of approximately $\frac{20n}{20 \cdot 5 + n}$. For a 1,000-player game, this reduces the requirement from 12 Mb/s to 670 Kb/s, making such a game feasible. Nonetheless, two important problems remain: First, some players may not have enough capacity to support even this modest average requirement. Second, player attention is not uniform: a player in whom many others are interested needs to send frequent updates to all of them.

Donnybrook addresses these problems by having peers with spare bandwidth assist peers without enough. The theoretical average bandwidth requirement described above is modest, so jointly optimizing update and guidance dissemination is not crucial. Furthermore, the needs of update and guidance dissemination are different, primarily since update receiver sets are far more dynamic. Therefore, we address them separately by allocating a fraction of aggregate capacity to each message type. Updates are most important since they determine the fidelity of objects players are paying attention to. Thus, each machine first reserves 200 Kb/s to send updates directly to those interested in them, which is enough in the common case for interest sets of size 5. As mentioned above, this will not be sufficient when many others are interested in a player, so a number of nodes with substantial remaining capacity join a "forwarding pool" to forward for these players. The remaining capacity in the system is used to disseminate guidance. The next two sections describe these two dissemination mechanisms in detail.

4. DISSEMINATING UPDATES

Each peer is essentially the source of a multicast group. Players who have that peer in their interest set are members

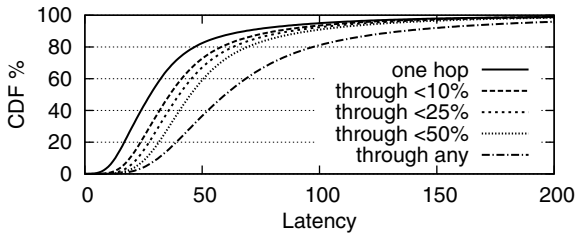

Figure 1: Effect of forwarding on latency, using latency model from §7.1.2. "Through $< n$%" means that messages go through a random forwarder whose average latency is below the nth quantile. Note that these values do not include queuing delay, and thus give probabilities higher than will be seen in practice.

of this group, called the peer's *subscriber set*. A subscriber set may be large, since even though each interest set is only five players, a particularly interesting player may be in as many interest sets as there are players. Our main challenge is how to allocate capacity to peers who do not have enough capacity to send updates to their subscriber sets directly.

Traditional approaches have nodes with spare capacity (etc.) [2, 11]. However, interest sets change rapidly, making it difficult to amortize the cost of building and optimizing multicast trees. Thus, Donnybrook has *sources* rather than subscribers dictate the structure of multicast trees. To avoid the need for complex coordination of forwarding resources between different groups, we do not build long-lived trees at all. Instead, each source independently builds a new forwarding tree each frame in a probabilistic fashion. This is possible because our modest bandwidth demands leave much capacity to spare, so we do not have to maximize its utilization. Although our approach limits scale, as it requires the source to know all its subscribers, it suffices for groups with hundreds of peers.

4.1 Forwarding pool

To forward updates, we use a *forwarding pool*, i.e., a set of machines with good connectivity properties and total rate limits sufficient to meet the aggregate forwarding needs of all sources. When a machine requires more outbound bandwidth within a frame than it can supply itself, it randomly chooses a subset of the forwarding pool that in aggregate is capable of supplying enough capacity. It then sends forwarding requests to those machines, each including a message and a set of recipients.

Determining membership and capacity. A machine considers itself part of the forwarding pool if its average latency to other nodes is below a certain threshold T_l and its rate limit is above a certain threshold T_r. The matchmaking service selects values for T_l and T_r at the start of a game as follows. It chooses T_l to exclude nodes above the 25th quantile of average latency; Figure 1 shows how restricting forwarding to well-connected nodes substantially increases the probability of delivering a message on time. Given T_l, it chooses T_r such that the total capacity in the forwarding pool is sufficient to satisfy all forwarding demands.

Each member of the forwarding pool computes the capacity it contributes as its rate limit minus 200 Kb/s, an amount reserved for local use.

Advertising capacity. Due to players leaving the game, and due to changes in rate limit and latency at a node, machines may enter and leave the forwarding pool, or change

their contributed capacity. To keep sources up to date about these changes, forwarding pool members *advertise* their capacity periodically. They do this by including a byte in their guidance specifying how many KB per frame they can forward. As guidance is distributed each second to each player, sources quickly learn of changes to the forwarding pool. A node can have slightly outdated information about forwarding pool membership, but the consequences of this are minor. Since it forwards each update through a random forwarder each frame, it is unlikely to send many messages to a departed forwarder before timing out its membership in the forwarding pool.

For two reasons, pool members only advertise half their pool capacity. First, because sources independently choose forwarding pool subsets each frame, they sometimes collide. Letting two sources use the same forwarder without overflowing decreases the likelihood a forwarding pool member must drop forwarding requests. Second, messages forwarded through a forwarding pool member take two hops before reaching their destinations, and thus cannot tolerate much queuing delay. Using all the capacity of a forwarder would lead to ∼50 ms of queuing delay at its outbound link.

Collisions. We could prevent multiple sources simultaneously choosing the same forwarder by coordinating allocation of forwarders to sources. However, there are three disadvantages to this approach. First, it cannot deal with unpredictable spikes in bandwidth demand, such as due to large update messages or sudden increases in subscriber set size. Second, coordination incurs protocol overhead and makes the system subject to failure of the coordination point. Third, we can achieve its main effect, reducing collision likelihood, simply by allocating more capacity than necessary to the forwarding pool. §7.5 shows that this does not substantially affect scalability since the bandwidth needed for the forwarding pool is a small fraction of the bandwidth needed for guidance.

Dynamic threshold selection. As nodes depart, the aggregate need for the forwarding pool drops along with the capacity of the forwarding pool. However, it may happen that nodes with high rate limits leave preferentially, leaving the system without sufficient capacity. Fortunately, each node has sufficient knowledge to locally detect this condition: from guidance messages, it knows the total pool capacity and the number of players still in the game. When a node detects too little pool capacity, it lowers its local T_r value each frame. Thus, additional nodes will eventually join the forwarding pool, restoring the needed balance.

4.2 Practical details

Handling message loss. When possible, we avoid acknowledging messages to conserve bandwidth. For some types of messages, loss can eventually be detected by the sender or receiver, so we do not use acknowledgments.

Updates are delta-encoded with respect to the last two frames to enable recovery from a missing update. If two consecutive updates are lost, the subscriber sends a `nack` indicating that the next update must not be delta-encoded. To avoid nack implosion, i.e., the overwhelming of sender inbound capacity when a burst loss causes all its subscribers to send simultaneous `nack`s, we have each subscriber send its `nack`s via the last forwarder for the corresponding source. This forwarder aggregates `nack`s for one frame and transmits them to the source.

NATs. Many player machines are behind NATs which can make inbound communication difficult. However, since we use UDP to deliver updates, a matchmaking service can help establish connection mappings between each pair of hosts at the start of a game using a protocol such as STUN [39]. Some NATs tear down these mappings after periods of inactivity, but we found that the overhead of sending necessary keep-alive traffic is minimal and impacts our performance results negligibly. Some "strict" NATs do not support STUN-like approaches [26]. We exclude players using such NATs, as p2p games typically do. We could support them by forwarding traffic between them, but this is future work.

5. DISSEMINATING GUIDANCE

Disseminating guidance is less challenging than disseminating updates since group membership changes rarely: each player always sends guidance to all others. We again use a subset of nodes as forwarders, to assist nodes with insufficient capacity. A side benefit is that by sending all guidance through forwarders, we can aggregate messages from many senders, and thereby reduce bandwidth for packet headers.

The central challenge is how to schedule the forwarding of guidance to make doppelgängers behave most realistically. Recall that each guidance message contains a prediction of what the player will do during the next second, such as where they will move. Our user study suggests that predictions of one second work well (see §6). Since a prediction is only valid until one second after its generation, each node must ensure that fresh guidance arrives before this time. Moreover, since it can take ∼1 second to carry out the predicted actions, e.g., moving to a target location, guidance must arrive with enough time for a doppelgänger to act on it.

5.1 Forwarder selection

Each machine can compute its fan-in, i.e., the number of sources it can forward for, as $F = \lfloor (pb - pr - hn)/ns \rfloor$, where s is guidance size, p is guidance period, b is the rate limit, r is bandwidth set aside for non-guidance, n is player count, and h is header size. In Donnybrook, $s = 50$ bytes, $p = 850$ ms (see §5.2), $r = 200$ Kb/s (see §4), and $h = 44$ bytes [29]. An exception is that forwarding pool members set $F = 0$: They do not forward guidance so they can devote as much capacity as possible to the forwarding pool.

Each machine with $F > 0$ acts as a forwarder. It always forwards at least for itself; if $F > 1$ it can forward for others as well. If $F \leq 0$, it must use as a forwarder a machine with $F > 1$. At the outset of the game, the matchmaking service assigns each source a forwarder such that no forwarder is assigned more sources than its fan-in. If such an assignment is not possible, the game is not feasible in Donnybrook. We evaluate the scales at which games are feasible in §7.2.

Every second, every forwarder sends one guidance packet to each other machine. This packet contains, for each source using it as a forwarder, the latest guidance it has received from that source. A forwarder determines its set of sources simply by observing from which machines it receives guidance-forwarding requests.

5.2 Transmit frequency

We now consider how often a source should send guidance to its forwarder. Define the *age* of guidance to be the time since it was generated. Figure 2(a) illustrates that the maximum age of guidance in use is the sum of t_1, the source-to-forwarder latency, t_2, how long the guidance waited at the

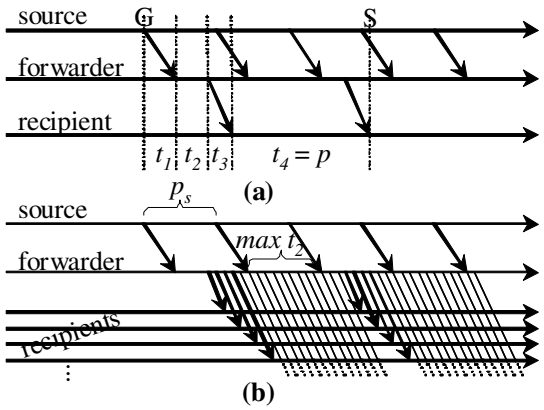

Figure 2: (a) The recipient uses guidance generated at time G until time S, so it uses guidance as old as $t_1 + t_2 + t_3 + t_4$ (see text). (b) Since the forwarder spends most of its bandwidth forwarding guidance, it is likely some receiver will see t_2 be the source's sending period p_s.

forwarder, t_3, the forwarder-to-recipient latency, and t_4, the guidance period p. Figure 2(b) shows that because the forwarder spends most of its bandwidth forwarding guidance to many recipients, it is likely some receiver sees t_2 equal to the source's sending period p_s.

Since we want to keep age under 1 sec, and we expect $t_1 + t_3 \approx 100$ ms, we want $t_2 + t_4 = 900$ ms. By using the smallest possible p_s, one frame or 50 ms, we maximize the forwarder period, which in turn minimizes forwarder bandwidth and thus maximizes scale. In addition, this minimizes t_2, so receivers get guidance with as much time as possible to realistically move doppelgängers to their predicted positions at the time of guidance expiry. This explains why we use $p = 850$ ms.

5.3 Dealing with churn

As the game progresses, some machines may leave the game, or their rate limits may change. Thus, sources need to find new forwarders when their current forwarders can no longer serve them.

Active probing is unnecessary for a source to detect its forwarder's failure, since forwarders periodically send guidance packets to everyone. If a few periods pass with no guidance packet from its forwarder, the source should choose a new one; we use a timeout of 3 seconds. If a forwarder's rate limit drops and it can no longer accommodate a source, it notifies the source immediately. Probing is also unnecessary for a forwarder to detect a source's failure. If a source stops sending a forwarder guidance, the forwarder drops it.

To choose a new forwarder, a source needs to know which machines have spare fan-in. To disseminate this information, a forwarder includes a bit in each forwarded guidance packet that indicates if it can support an additional source. A source needing a new forwarder selects a random forwarder that has recently advertised available fan-in. If forwarders also send these advertisement bits to the matchmaking service, it can admit new players when there is sufficient capacity. However, we have not evaluated how often new players may be refused.

6. EVALUATION: USER EXPERIENCE

We hypothesize that using low-fidelity replicas of players outside a player's interest set will not reduce the fun of game play. We validate this hypothesis with the following user study, that considers three scenarios: LoBW-IS, a low-bandwidth setting using interest sets; LoBW, a low-bandwidth setting using the standard p2p approach; and HiBW, a high-bandwidth setting. Our hypothesis makes two predictions: First, players will prefer LoBW-IS to LoBW, demonstrating that the trade-off of fidelity for bandwidth pays off. Second, players will enjoy LoBW-IS as much as HiBW, demonstrating that they do not mind the loss of fidelity. To study these scenarios, we modified Quake III so that it can run any of them. We needed under 4,000 lines of code to implement interest sets, guidance, and doppelgängers, and an additional ∼6,000 lines to change Quake III from a client-server to a p2p game.

6.1 Methodology

We approximate the following real-life scenario: When players select a public server to play on, they typically choose a server from a list; if they become dissatisfied with a server, they leave and try another. By this process, players implicitly express a preference between versions of the same game.

Procedure. We invite pairs of volunteers to play a game of Quake III. Each pair is told that there are two servers with different network characteristics, but we do not say how they differ. Some pairs are choosing between LoBW-IS and LoBW, and some pairs are choosing between LoBW-IS and HiBW. For each pair of players, we select a starting server at random, and allow them to switch servers whenever both players indicate a desire to switch. They may switch back and forth as often as they wish, and their game scores are preserved across each switch. After playing for a total of 15 minutes (a typical duration for such FPS games), they play a new 5-minute game on whichever version they played less. No switching is allowed during this follow-up game so they can make an informed comparison of the two games.

To simulate a larger game, we use 30 standard Quake III bots in addition to the two human players. Each bot or player is treated as a separate peer in the virtual network. All games are *death match* games, in which each player independently tries to score points by killing others. Each game uses the popular `q3dm17` map, which includes many typical FPS game features that impact interest sets and doppelgängers. For instance, it has long-range sniper weapons, close-range combat weapons, powerups, teleporters, and jump pads that enable players to "fly" long distances. In this map all players are in a single AOI, so AOI filtering would not reduce bandwidth requirements.

To highlight any perceptible inconsistencies in the game experience, we take several steps to encourage the human players to focus mostly on each other. For example, we make human players easy to identify, we make killing a human worth ten times more than killing a bot, and we encourage players to communicate verbally, through which they can reveal anomalies to each other.

Testbed. Each peer is executed by a virtual server on a single machine. We emulate a network between them. In LoBW and LoBW-IS, each peer, whether player or bot, has a rate limit of 108 kbps; HiBW has no rate limits. We choose 108 kbps because it enables a subscriber set size of four and a best-effort rate of one update per second in LoBW-IS, settings that we found satisfactory during play-testing. At this rate limit, a LoBW peer sends 5 updates to each player per second (in contrast to the nominal 20). Each peer manages its send

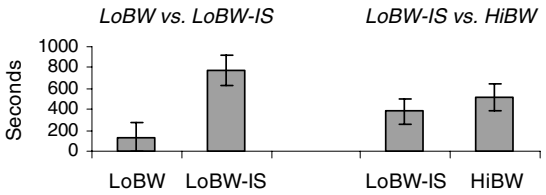

Figure 3: Total time players spend on each version. Error bars show 95% confidence intervals.

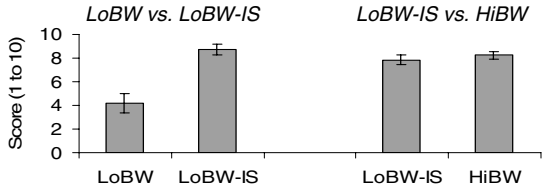

Figure 4: Average response to question, "How fun was this game on a scale of 1 (not fun) to 10 (extremely fun)?" Error bars show 95% confidence intervals.

rate to not exceed the rate limit, so there is no packet loss. The RTT between each peer is 20 ms.

The focus of this user study was not update distribution so we used games where bots' attentions are roughly uniformly distributed. Thus, each peer could send frequent updates directly to the four subscribers with highest interest and still satisfy most interest sets of size four. Our simulations in §7 use a more conservative interest set size of five to further improve user satisfaction.

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

Limitations. Our user study has two main limitations: it has only two human players, and it has only 32 players total. To mitigate the first limitation, we try to keep the human subjects focused on each other, so we can evaluate how well interest sets estimate focus on human players, not bots. We expect the second limitation to be minor in practice since human cognitive limitations do not slacken when more potential objects of attention are present. Although game play may change in such scenarios, game designers can tune our heuristics and increase the estimated interest set size so that it is more likely to include the true interest set.

6.2 Results

Two metrics show that interest sets preserve user satisfaction in games even in low-bandwidth environments:

Total time spent. Players are likely to play longer in a game that they find more enjoyable. Fig. 3 shows the average amount of time spent on each version in the two different experiment types. Given the choice between LoBW and LoBW-IS, players overwhelmingly choose to spend time on LoBW-IS. Anecdotally, this is due to the contrast between the natural movement of LoBW-IS's doppelgängers and the choppy and unrealistic movement caused by LoBW-IS's 200-ms update period. When choosing between LoBW-IS and HiBW, players spend slightly more time on HiBW, though this difference is statistically insignificant.

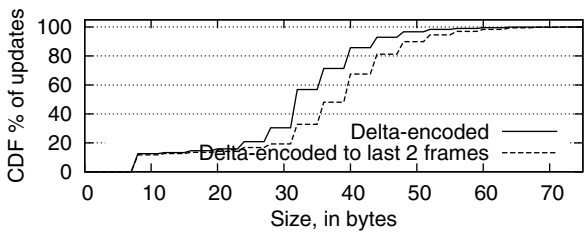

Figure 5: CDFs of update sizes for different levels of delta-encoding. Without delta-encoding, an update is 196 bytes.

Self-reported score. After the experiment, players rate their enjoyment of the two versions on a 1–10 scale. Fig. 4 shows the average scores given to each version in the two comparisons. On average, LoBW-IS is preferred by 4.5 points over LoBW, whereas the difference between LoBW-IS and HiBW is statistically insignificant.

7. EVALUATION: SCALE & PERFORMANCE

This section presents simulation results that evaluate Donnybrook under large-scale, wide-area-network conditions. First, we evaluate the battle scale Donnybrook allows us to achieve, and show that even at the highest scale, we achieve low guidance staleness and reasonable on-time update delivery rates (§7.2). We also show that we handle interest heterogeneity (§7.3) and churn (§7.4). We then conduct experiments to explain design decisions. We show the effectiveness of the forwarding pool and explain why we reserve 50 Kb/s per player for it (§7.5). We also justify our use of a forwarding pool by comparing to alternate approaches (§7.6). Finally, we explore the sensitivity of Donnybrook's scale to parameters that may differ in different games and environments, specifically capacity variability (§7.7) and required interest set size (§7.8).

7.1 Experimental Methodology

To evaluate the efficacy of Donnybrook for larger games, we developed a detailed simulator, described below.

7.1.1 Game Workloads

Existing fast-paced games rarely support more than 32 players, so it is not possible for us to obtain a real workload with hundreds of players. Instead we use the game workload generator developed by Bharambe *et al.* [4], which is based on the observed behavior of real Quake III players. Update sizes are drawn from the empirical distribution shown in Figure 5. To validate these workloads, we simulate a 32-player death match game like those in our user study. Figure 6 compares the subscriber set size distributions. Both the user study and death-match simulation distributions are approximately normal, though the subscriber sets are slightly more skewed in the user studies. Simulated capture-the-flag games, described below, make up some of the difference between our simulator and user study results, as can be seen in the figure.

To evaluate Donnybrook under skewed player attention distributions, we modify the generator to simulate capture-the-flag (CTF) games. Our modifications are based on Quake III's original code controlling how bots behave in CTF games, and on empirical CTF traces of real players. The structure of a CTF game is that players are divided into two teams and repeatedly attempt to capture the other

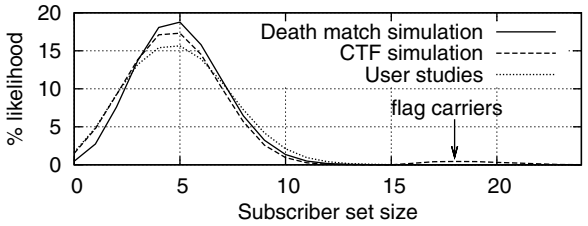

Figure 6: Comparison of subscriber set size distribution seen in user studies and in simulated games

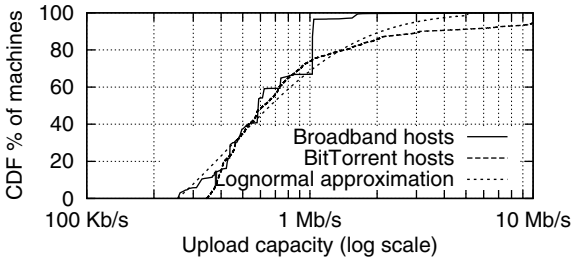

Figure 7: CDFs of peer upload capacities

team's flag. Thus, the players currently carrying the flags tend to be the focus of attention for many players, resulting in a highly skewed distribution of player attention. This is exemplified in Figure 6 by the second mode between 15 and 20. Note that unlike the mode around 5, this mode increases with the number of players in the game.

In our user studies, we observed that interest sets change frequently: on average, the set of 5 most interesting players changes 3.4 times per second. This corresponds to significant interest flux, 68% turnover per second. Our simulator captures this effect at a coarse level, producing rates of 2.6 (death match) and 2.9 (CTF) changes per second.

7.1.2 Network Model

In our games, each player is on a separate machine in a simulated network. The simulation uses empirical models of capacity, latency, and loss to represent what game players generally experience on the Internet.

Bandwidth. The primary factor affecting the feasibility of peer-managed epic battles is peers' upload capacity (see §2.2). To our knowledge, no upload capacity models exist for game clients, so we use two empirical distributions that we believe are representative: one of broadband hosts in the U.S. that have run a test to measure their capacity [7] and one of BitTorrent hosts [34] (Figure 7). The former is a conservative model that includes only home users, while the latter assumes that high bandwidth university and corporate sites can contribute as well. We impose a 4–6 Mb/s bandwidth cap on each peer in the latter model since it is unrealistic to expect higher rates to be acceptable, as we discussed in §2.2. Since these capacities are almost all much smaller than bottlenecks in the Internet's core, we do not model the bandwidth effects of cross traffic. In the sequel, we will refer to the broadband-host distribution as BB, and the BitTorrent distribution capped at n Mb/s as P2P-n.

We note that the second and third quantiles of both empirical distributions are similar and are well modeled by a log-normal distribution (with mean 1 Mb/s and a range of 256 Kb/s to 5 Mb/s), also shown in the figure. Therefore, we scale this analytic distribution to evaluate the sensitivity of our approaches to different distributions.

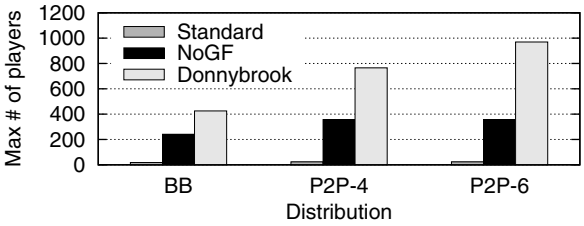

Figure 8: Maximum scale achievable by Donnybrook with various capacity distributions, compared to idealized models: Standard (no interest sets) and NoGF (no guidance forwarding)

Latency. For Donnybrook games, a matchmaking service would likely partition players into games based on geographical boundaries, to reduce internode latency and to combine players speaking the same language. To ensure large games, the geographical areas would have to be large. Thus, we emulate the inter-node RTT in an n-node game by drawing n nodes from the Halo 3 player data set [23] that are in the Western U.S., with longitude between 110° and 125° west. The mean, median, and standard deviation of inter-node RTT of this peer set are 81 ms, 64 ms, and 63 ms, respectively. Since most node pairs did not probe each other, we extrapolate RTT values using Vivaldi with a 3D-with-height metric space [12]. To capture latency variation due to cross traffic, we apply the model of Mukherjee [27].[1]

Loss. Finally, packet loss is important because its frequency determines how often updates must be retransmitted or otherwise recovered. Following the recommendations of Zhang et al. [42], we model loss using a two-state Gilbert model [17], setting loss rate to 1% and mean loss burst time to 100 ms.

7.1.3 Performance metric

As discussed in §2.1, lag is the most important network factor that affects player enjoyment and performance. Thus, the primary performance metric we measure is percentage of required updates that are received on time. An update is required if the player that generated it has been in the receiver's interest set for at least one frame, i.e., we estimate the receiver is interested in the player. Since round-trip times under 150 ms are generally recommended, and FPS games can cope with loss rates of 5–10% as long as updates are delivered within 100–150 ms [3, 23], we evaluate update delivery rates using 150-ms deadlines. Generally, delivery time distributions are shaped similarly to the latency distributions in Figure 1, so when 96% of updates arrive within 150 ms, over 90% of updates arrive within 100 ms and performance should be acceptable.

7.2 Scalability

We start by evaluating the effect of Donnybrook's techniques on battle scalability. We consider a scale n to be achievable if a randomly selected set of n machines has probability $\geq 99.5\%$ of satisfying the initial feasibility test performed by the matchmaking service. For Donnybrook, this feasibility test checks whether sufficient capacity exists for

[1]The instantaneous one-way delay for each pairwise connection is drawn from an offset gamma distribution, whose mean is set to half of the mean RTT obtained from the above procedure. Based on measurements and regression analyses of our own, we set the distribution such that 75% of the mean delay comes from the constant offset and 25% comes from the gamma distribution. The order, i.e., shape parameter, of the gamma distribution is 0.1.

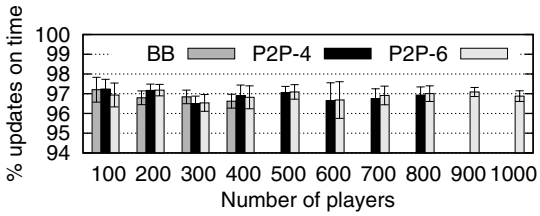

Figure 9: Updates delivered within 150 ms at various scales with various capacity distributions. No results shown for scales where no run passed the initial feasibility test (500+ for BB, 900+ for P2P-4). Error bars show 95% confidence intervals.

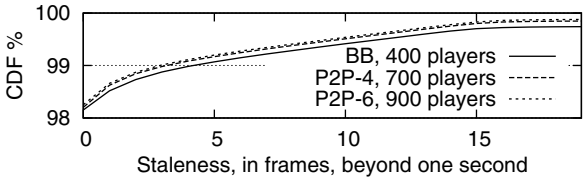

Figure 10: How often a player is operating on stale guidance from another player.

forwarding guidance after setting aside capacity for the forwarding pool.

Figure 8 shows the results: Donnybrook's maximum scale is 425 for BB, 765 for P2P-4, and 969 for P2P-6. For comparison, we also show scale for optimistic models of simpler approaches. The Standard model represents architectures like Colyseus [4], where each player sends updates to each other player in the battle every frame. Architectures that rely on AOI filtering to limit bandwidth requirements [19, 21, 22, 38] degenerate to the Standard model when all players are in the same AOI. The NoGF model represents the use of interest sets but not guidance forwarding. NoGF achieves 13–16 times more scale than Standard, since it sends 20 times less data to most but not all peers. Donnybrook achieves 2–3 times yet more scale by allowing high-capacity machines to forward for low-capacity ones. This makes scale constrained by the mean capacity rather than the minimum capacity.

Of course, the initial feasibility test is insufficient proof that Donnybrook can achieve a certain scale. Its performance must also be reasonable. To test this, we run 30-minute CTF games of Donnybrook at various scales and evaluate performance. Since these simulations are time-intensive, we evaluate only scales that are multiples of 100 players, and run 11 games each. Figure 9 shows the results, which demonstrate that these scales are indeed achievable. Performance is acceptable at all scales, with deadline miss rates of only ∼3–4%. Indeed, we can tell that performance is even reasonable when we exceed the 99.5%-likely achievable scale, since many of our runs beyond this scale were feasible. 11/11 of the P2P-4 runs with 800 players, and 9/11 of the P2P-6 runs with 1000 players, were feasible.

We also need to show that guidance is generally delivered before the previous guidance has expired. Figure 10 shows a graph of guidance staleness for the three distributions at maximum scale. The primary causes of stale guidance are high latency and packet loss. We see that a doppelgänger operates on guidance that is at least one frame too stale only about 1.5% of the time. Furthermore, used guidance is more than 1 sec stale only about 0.25% of the time. Hence, Donnybrook's guidance dissemination strategy effectively meets latency bounds, despite its use of two-hop forwarding.

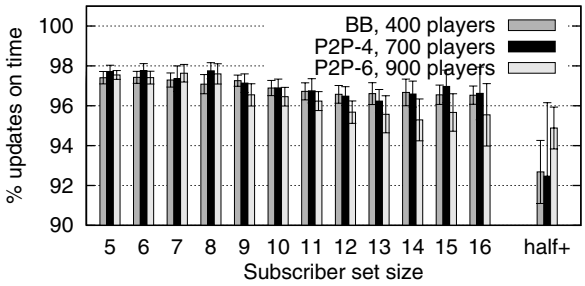

Figure 11: Updates delivered within 150 ms, as a function of the subscriber set size of the source. Error bars show 95% confidence intervals.

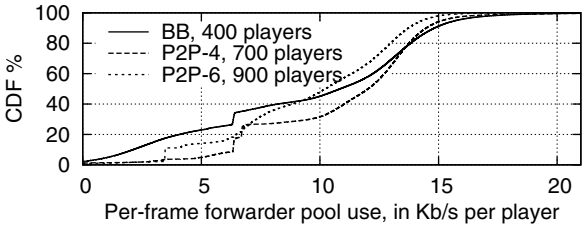

Figure 12: How often various amounts of bandwidth were requested from the forwarding pool

We conclude that Donnybrook can support at least 400 players with BB, 700 with P2P-4, and 900 with P2P-6.

7.3 Effect of subscriber set size

One of our goals is to quickly deliver updates even from players with high subscriber set sizes, despite those players' capacity limitations. Figure 11 shows how many updates are delivered within 150 ms for different subscriber set sizes. We observe that even when subscriber set sizes are massive, constituting at least half the players, sources can still deliver well over 90% of their updates within 150 ms. Slightly more updates take over 150 ms as subscriber set size increases due to increased queuing delay and the fact that more messages are sent via multi-hop paths.

7.4 Churn

To simulate churn, we assume each node uses an exponential distribution to independently decide when to leave. We simulated various churn rates, corresponding to probabilities of 10–50% of leaving during the 30-minute game. In all our results (not shown), performance with churn is never less than performance without churn by more than the width of the 95% confidence interval, indicating our techniques for handling churn are effective. Indeed, often results with churn are slightly better than without it. This is because during periods with fewer players in the game, guidance bandwidth demand is lower and thus queuing delay is slightly reduced.

7.5 Forwarding pool

A key question about the forwarding pool is how much bandwidth to set aside for it. To answer this, we evaluate (1) how much bandwidth is generally needed in a frame, and (2) how forwarding pool size affects scalability.

To address the first issue, we simulate several large-scale games and record the aggregate bandwidth requested from the forwarding pool each frame. Figure 12 shows the results. The requested amount varies but is rarely more than

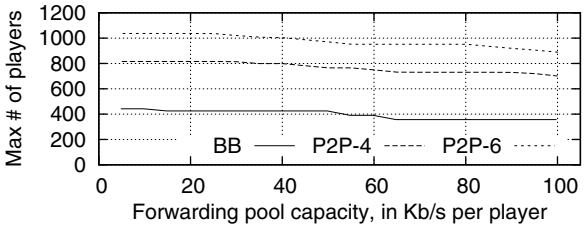

Figure 13: Achievable scale as a function of total forwarding pool capacity, expressed as Kb/s per player

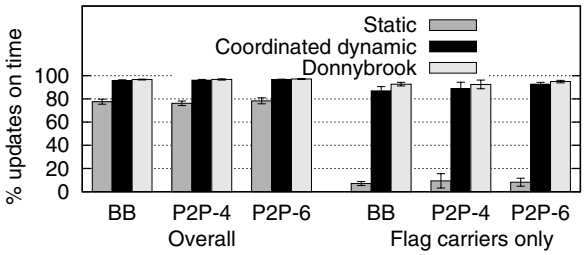

Figure 14: Comparison of Donnybrook's forwarding pool to alternative approaches. Player count is 400 for the BB distribution, 700 for P2P-4, and 900 for P2P-6.

20 Kb/s per player. The maximum ever requested is 49 Kb/s per player. These values are low because the forwarding pool is mainly used by players with high subscriber set sizes, and in our CTF games there are only two such players, the flag-carriers. Even if every player in the game is interested in both flag carriers, and both send full updates to every player, their aggregate requirement is only 38 Kb/s per player.

To address the second issue, Figure 13 shows how the maximum scale varies according to the per-player bandwidth set aside for the forwarding pool. We compute maximum scale, as before, as the value that allows $\geq 99.5\%$ of player groups to satisfy the initial feasibility test. We see that the more capacity we set aside for the forwarding pool, the less capacity is available for guidance dissemination, which in turn reduces scale. However, we also see that this effect is minor. Since scale is relatively insensitive to the bandwidth reserved for forwarding pool use, we conclude we can comfortably allocate a generous amount. Thus, we set aside 50 Kb/s per player in the forwarding pool, providing ample bandwidth and thus keeping down the probability that sources' random forwarder selections will overload any given forwarder.

To verify the necessity of having any bandwidth at all in the forwarding pool, we conducted simulations without any forwarding pool. We found, as expected, that this rendered performance unacceptable, often with fewer than 80% of updates being delivered on time.

Finally, an important goal of the forwarding pool is to ensure *fairness* among players, in particular not to discriminate against players with low rate limits. We find that the correlation coefficient between rate limit and fraction of updates delivered on time is, on average very low, 0.0002, which indicates a high degree of fairness.

7.6 Alternate update forwarding strategies

In this section, we evaluate the forwarding pool against two alternate approaches. One, static assignment, allocates forwarders to sources at the outset of the game, similarly to how supernodes are chosen in other p2p systems. The second, coordinated dynamic assignment, uses a leader node to allocate forwarders to sources based on dynamically-changing subscriber set size [14].

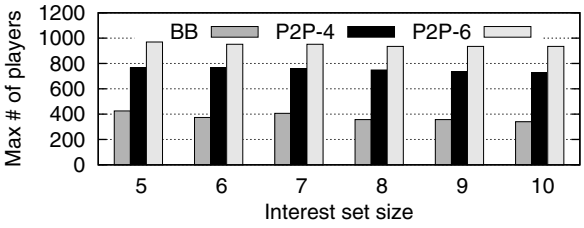

Figure 15: Maximum scale as a function of interest set size for various capacity distributions

Figure 14 shows the results. We observe that the static assignment approach produces unacceptable update delivery performance at high scales, especially for players with high subscriber set sizes. A single machine acting as a forwarder, even if it has as much as 6 Mb/s, has difficulty serving as a forwarder for a set of sources when one or more of them attracts a large number of subscribers.

We also observe that the coordinated dynamic approach produces worse performance than Donnybrook. This result is surprising since coordinating forwarding capacity assignment should reduce contention for the forwarding pool However, in practice, three effects lead to the coordinated approach suffering. First, and foremost, the coordinated approach seeks unused capacity anywhere it is present, and thus often uses forwarders with lower capacity or higher average latency than machines in the forwarding pool. Lower capacity means it takes longer for packets to drain from their outbound queues, causing increased queuing delay. Second, the coordinated approach requires overhead to maintain coordinated information about where capacity is available and needed. Finally, forwarding pool contention is rare, making the advantage of coordination of minor value.

7.7 Capacity variation

To determine the effect of inter-player capacity variability, we conducted evaluations using the log-normal approximation of our capacity distributions. That approximation varies from 256 Kb/s to 5.1 Mb/s and has a mean of 1 Mb/s. By changing the variance in our model but keeping the mean the same, we generated two other distributions, one varying less, from 384 Kb/s to 3.6 Mb/s, and one varying more, from 128 Kb/s to 8.0 Mb/s. We find that as variance increases, the maximum scale achievable goes from 646 to 683 to 765. Greater variation enables greater scale, even with the same mean, because using high-bandwidth nodes provides more opportunity for packet coalescing. Performance results (not shown) indicate that we achieve acceptable performance on all three distributions.

7.8 Interest set size variation

Some games may place less cognitive burden on players and allow them to focus on more than five other players at a time. Also, limitations of our user study prevent us from definitively concluding that a set of five is sufficient for all games. Therefore, it is useful to understand what happens when interest set sizes are larger than five. To evaluate this, we consider interest set sizes ranging from five to ten. For each increase of one in interest set size, we reserve an additional 12 Kb/s on each machine for distributing updates, reducing the amount available for forwarding guidance or the forwarding pool.

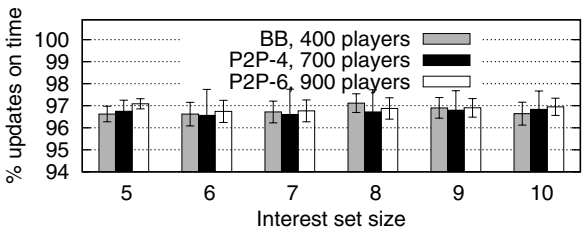

Figure 16: Performance as a function of interest set size for various capacity distributions and player counts. Error bars show 95% confidence intervals.

Figure 15 shows the effect on scale of increasing interest set size, largely due to the additional reserved capacity. We see that interest set sizes of up to ten do not substantially reduce scale. Furthermore, simulation results, shown in Figure 16, indicate that with the additional capacity reservation, performance does not change significantly even with interest set size up to 10. We conclude that Donnybrook can still achieve nearly the same high scale.

8. RELATED WORK

8.1 Interest filtering

Efforts to improve the quadratic scaling of traditional game designs usually leverage the fact that each player has a limited *area of interest* (AOI), i.e., he is interested in only a subset of the entire game state, typically his local area or field of view. Most p2p AOI designs perform interest filtering by partitioning the game world into disjoint regions [16, 21, 22, 38]. Colyseus [4] and VON [19] allow peers to specify arbitrary AOIs.

AOI filtering reduces bandwidth demand to the extent that players have limited proximity to each other. In an epic battle, however, everyone will be in the same AOI, and AOI filtering will not reduce bandwidth demands. For instance, despite Colyseus's scalable object discovery, it will still require a player to send updates every frame to every other player who is near his objects. Hence, bandwidth requirements remain quadratic in the number of players in the battle. Our interest set design goes beyond AOI filtering by allowing clients to specify their *degree* of interest in nearby objects, and by maintaining object replicas with different fidelity based on interest level.

8.2 Variable object fidelity

Two techniques that reduce the required frequency of object state updates are predictive contract mechanisms (PCM) [20] and multi-resolution simulation [18]. In a PCM, the sender and receiver use a shared model to predict the object's state, and the sender only sends a state update when the predictor's error exceeds a threshold. The most common PCM uses *dead reckoning*, extrapolating an object's position based on past position, velocity, and sometimes other moments [32]. Other predictors include a hybrid dead-reckoning/shortest-path predictor [24] and a neural network [25]. Like PCMs, Donnybrook also predicts future behavior and uses this to reduce bandwidth demand.

Multi-resolution simulation schemes, when used in the context of distributed interactive simulation [13], provide multiple implementations for each object, with varying degrees of resolution and simulation cost. This resembles Donnybrook's two representations of a remote player: the standard representation and the doppelgänger.

The most important difference between these techniques and Donnybrook is that while the former strictly bound error and produce unpredictable and modest bandwidth reductions, Donnybrook reduces bandwidth by a large, predictable factor while allowing unbounded error. This is tolerable because we only reduce fidelity for objects the receiver is not paying attention to. Our techniques can cause inconsistent views across players, but we comfortably ignore this divergence because our user studies confirm that the game remains enjoyable.

8.3 Overlay multicast

Update delivery requires four properties from a multicast scheme: (1) a strict bound on end-to-end latency, (2) support for very frequent group membership changes, (3) support for heterogeneity in client capacity, and (4) scalability with the number of groups and total membership.

Structured (DHT-based) multicast designs [8, 9] allow nodes to join quickly and minimizes control overhead. Thus, such schemes handle (2) and (4) well. However, it is very difficult to optimize latency or handle node heterogeneity effectively [36]. At the other end, unstructured approaches [2, 10, 30] allow clients to join anywhere and try to optimize the tree structure. As a result, they are better at supporting (1) and (3) but worse at (2) and (4) due to control overhead. Our approach is unstructured, but, in contrast to most schemes, has the *source* rather than the clients control tree structure. This minimizes control overheads and can still scale to 900-player games. CoopNet [30] also uses source-controlled trees, but requires Multiple Description Coding to encode video frames over multiple trees to handle churn and packet loss. However, game updates are too small relative to packet headers to be efficiently encoded in this way and CoopNet does not address issues in scaling large numbers of partially overlapping groups.

9. CONCLUSIONS AND FUTURE WORK

In this paper, we presented Donnybrook, a system that enables peer-managed, large-scale, high-speed games by supporting epic battles involving hundreds of players. Donnybrook supports such battles with three essential elements: First, it leverages human cognitive limitations to preferentially spend bandwidth capacity on information most important to players using interest sets. As a consequence, the volume of frequent updates scales only linearly with the number of players rather than quadratically, making large scales feasible. Second, Donnybrook uses a delay-sensitive forwarding mechanism to disseminate the remaining necessary all-to-all information, so that scale is not limited by the node with the smallest capacity. Third, Donnybrook uses a latency-sensitive, uncoordinated dynamic allocation system to forward frequent updates, allowing it to support extreme heterogeneity in player popularity and capacity.

Although this paper discussed the use of interest sets and doppelgängers in the context of p2p games, these techniques should also be useful in client/server games. The bandwidth a server requires also scales quadratically with the number of players in an AOI, so our techniques can reduce the cost of running such a game by reducing this requirement. Evaluating this approach is future work.

We are currently working on an Internet deployment of a large-scale game using Donnybrook, to learn more about the benefits and limitations of our approach. While our user study allowed us to evaluate the effectiveness of interest sets

in a controlled, experimental setting, it may have missed factors that will only manifest in a true large-scale game. For instance, large scale may reveal properties of interest sets we were unable to observe in smaller-scale games, necessitating more sophisticated heuristics or larger interest set sizes. Also, we may discover how players try to gain unfair advantage from Donnybrook's new techniques and what is necessary to defend against such cheating. Furthermore, an Internet deployment will more accurately reflect the networking characteristics of game-player communities.

10. ACKNOWLEDGMENTS

The authors wish to thank Frank Uyeda for his work in implementing guidable AI in our Quake III prototype. He also tested early prototypes and helped manage the user study. We also thank the anonymous reviewers.

11. REFERENCES

[1] Akella, A., Seshan, S., and Shaikh, A. An empirical evaluation of wide-area Internet bottlenecks. In *SIGCOMM* (Oct. 2003), pp. 316–317.

[2] Banerjee, S., Lee, S., Bhattacharjee, B., Srinivasan, A., and Braud, R. Scalable resilient media streaming. In *NOSSDAV* (June 2004).

[3] Beigbeder, T., Coughlan, R., Lusher, C., Plunkett, J., Agu, E., and Claypool, M. The effects of loss and latency on user performance in Unreal Tournament 2003. In *NetGames* (Aug. 2004), pp. 144–151.

[4] Bharambe, A., Pang, J., and Seshan, S. Colyseus: A distributed architecture for interactive multiplayer games. In *NSDI* (May 2006).

[5] Blizzard Entertainment. Battle.net info. http://www.battle.net/info/faq.shtml, 2007.

[6] Blizzard Entertainment. WoW PvP battlegrounds. http://www.worldofwarcraft.com/pvp/battlegrounds, 2008.

[7] Broadband Reports. Broadband reports speed test statistics. http://www.dslreports.com/archive, Sept. 2007.

[8] Castro, M., Druschel, P., Kermarrec, A.-M., Nandi, A., Rowstron, A., and Singh, A. Splitstream: High-bandwidth multicast in a cooperative environment. In *SOSP* (2003).

[9] Castro, M., Druschel, P., Kermarrec, A.-M., and Rowstron, A. Scribe: A large-scale and decentralized application-level multicast infrastructure. *IEEE JSAC 20*, 8 (Oct. 2002).

[10] Chu, Y.-H., Rao, S. G., Seshan, S., and Zhang, H. A case for end system multicast. *IEEE JSAC 20*, 8 (2002).

[11] Cowan, N. The magical number 4 in short-term memory: A reconsideration of mental storage capacity. *Behavioral and Brain Sciences 24*, 1 (2001).

[12] Dabek, F., Cox, R., Kaashoek, F., and Morris, R. Vivaldi: A decentralized network coordinate system. In *SIGCOMM* (Aug. 2004), pp. 15–26.

[13] DIS Steering Committee. The DIS vision: A map to the future of distributed simulation, ver. 1. IST-SP-94-01, 1994.

[14] Douceur, J., Lorch, J., and Moscibroda, T. Maximizing total upload in latency-sensitive P2P applications. In *SPAA* (June 2007), pp. 270–279.

[15] GameZone. Blizzard Entertainment Announces World of Warcraft Release Plans for Europe. http://pc.gamezone.com/news/11_24_04_09_39AM.htm, 2004.

[16] Gautier, L., and Diot, C. Design and evaluation of MiMaze, a multi-player game on the Internet. In *IEEE Multimedia Systems Conference* (July 1998).

[17] Gilbert, E. N. Capacity of a burst-noise channel. *The Bell System Technical Journal 39* (1960).

[18] Hamilton, J. A., Nash, D. A., and Pooch, U. W. *Distributed Simulation*. CRC Press, 1997.

[19] Hu, S.-Y., Chen J.-F., and Chen, T.-H. VON: a scalable peer-to-peer network for virtual environments. *IEEE Network 20*, 4 (July/Aug. 2006).

[20] IEEE. IEEE standard for distributed interactive simulation—application protocols, Sept. 1995. IEEE Standard 1278.1-1995.

[21] IEEE. IEEE standard for modeling and simulation high level architecture (HLA), Sept. 2000. IEEE Standard 1516-2000.

[22] Knutsson, B. et al. Peer-to-peer support for massively multiplayer games. In *INFOCOM* (July 2004).

[23] Lee, Y., Agarwal, S., Butcher, C., , and Padhye, J. Measurement and estimation of network QoS among peer Xbox 360 game players. In *PAM* (Apr. 2008).

[24] McCoy, A., McLoone, S., Ward, T., and Delaney, D. Dynamic hybrid strategy models for networked multiplayer games. In *ECMS* (June 2005), pp. 727–732.

[25] McCoy, A., Ward, T., McLoone, S., and Delaney, D. Multistep-ahead neural-network predictors for network traffic reduction in distributed interactive applications. *TOMACS 17*, 4 (Sept. 2007).

[26] Microsoft Corporation. Xbox LIVE website. http://www.xbox.com/en-US/live/, 2007.

[27] Mukherjee, A. On the dynamics and significance of low frequency components of Internet load. *Internetworking: Research and Experience 5* (1994).

[28] Ohio University. Ohio University announces changes in file-sharing policies. http://www.ohio.edu/students/filesharing.cfm, 2007.

[29] O'Neill, V. Adding creamy nougat and a crisp candy coating to the network: XRNM and QNet. *XNA GameFest* (Aug. 2007).

[30] Padmanabhan, V., and Sripanidkulchai, K. The case for cooperative networking. In *IPTPS* (Mar. 2002).

[31] Pang, J., Uyeda, F., and Lorch, J. Scaling peer-to-peer games in low-bandwidth environments. In *IPTPS* (Feb. 2007).

[32] Pantel, L., and Wolf, L. C. On the suitability of dead reckoning schemes for games. In *NetGames* (Apr. 2002), pp. 79–84.

[33] Paul, R. More universities banning Skype. http://arstechnica.com/news.ars/post/20060924-7814.html, 2006.

[34] Piatek, M., Isdal, T., Anderson, T., Krishnamurthy, A., and Venkataramani, A. Do incentives build robustness in BitTorrent? In *NSDI* (Apr. 2007), pp. 1–14.

[35] Pittman, D., and Dickey, C. G. A measurement study of virtual populations in massively multiplayer online games. In *NetGames* (Sept. 2007).

[36] Rao, S., Bharambe, A., Padmanabhan, V., Seshan, S., and Zhang, H. The impact of heterogeneous bandwidth constraints on DHT-based multicast protocols. In *IPTPS* (Feb. 2005).

[37] Robson, J. G., and Graham, N. Probability summation and regional variation in contrast sensitivity across the visual field. *Vision Research 21*, 3 (1981).

[38] Rosedale, P., and Ondrejka, C. Enabling player-created online worlds with grid computing and streaming. *Gamasutra* (Sept. 2003).

[39] Rosenberg, J., Weinberger, J., Huitema, C., and Mahy, R. STUN - Simple Traversal of User Datagram Protocol (UDP) Through Network Address Translators (NATs). RFC 3489 (Proposed Standard), Mar. 2003.

[40] Sony Online Entertainment. PlanetSide FAQ. http://planetside.station.sony.com/faq.vm, 2008.

[41] YVG Staff. Video game sales break records. http://us.i1.yimg.com/videogames.yahoo.com/feature/video-game-sales-break-records/1181404, Jan. 2008.

[42] Zhang, Y., Duffield, N., Paxson, V., and Shenker, S. On the constancy of Internet path properties. In *IMC* (Nov. 2001).