Scaling Peer-to-peer Multiplayer Games with Donnybrook

Jeffrey Pang — *Carnegie Mellon*
Frank Uyeda — *UC San Diego*
Jacob R. Lorch, John R. Douceur — *Microsoft Research*

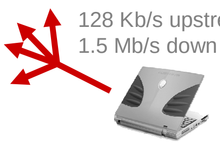
Our goal: Dramatically increase the scale of peer-to-peer shooter games

Why massive-scale P2P games?

- Massively multiplayer games growing in popularity
- Leveraging participants' machines has advantages
  - Reduces subscription costs, dependence on infrastructure
  - Allows scaling to arbitrary numbers of clients
- P2P because using one peer as a server limits game scale

The outbound bandwidth problem

- In large games, players can see many others
  - You send position updates to everyone who can see you
- Residential broadband is asymmetric
- Insufficient capacity to send updates at preferred frequency



128 Kb/s upstream,
1.5 Mb/s down

Example (Quake III)

Update rate:	*20 updates/sec*
Update size:	*100 bytes*
Bandwidth needed per peer:	*16 kb/s*
Supportable peers at 128 kbps:	*Only 8!*

Donnybrook design principles

- Some updates very time-critical, but only to one peer
  - Donnybrook component: Pairwise rapid agreement
- Player attention bounded when in a crowd
  - Inconsistencies more likely to be noticed in certain objects
  - Donnybrook component: Focus set
- For out-of-focus objects, realism more important than accuracy
  - Donnybrook component: Guidable AI

Pairwise rapid agreement

- Scenario: Object A modifies another object B
  - Alice does damage to Bob
  - Alice dies, increasing Bob's score
  - Alice picks up an item or opens a door
- Goal: Modification consistent and applied quickly
  - If Alice brags about hitting Bob on voice chat, he should know what she's talking about
- Approach: Essentially, asynchronous RPC
  - Peer managing object A sends update to B's peer
  - Object B updated immediately
  - Scalable: 1-to-1 communication, not 1-to-n

Focus set

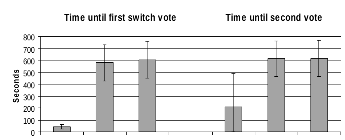
- When inconsistent states are likely to be noticed, infrequent updates are bad
  - Alice intently watching, following, and/or shooting Bob
  - Bob driving a vehicle Alice is riding in
- Each peer reserves fraction of upload bandwidth to send updates each frame to its focus set
  - Set is the *n* players most interested in its avatar's state
  - *n* is constant (~4 or 5), so bandwidth is constrained
- How focus set is chosen
  - Each peer A computes *attention value* heuristic for each other peer B, estimating A's interest in B's state
  - Peers piggyback these values on update messages
  - Peer B selects *n* highest such values for its focus set

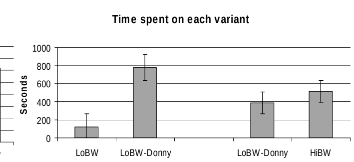
Guidable AI

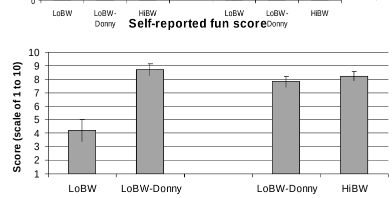
- Limited bandwidth for peers not in focus set
  - Not a lot to begin with, then focus set takes most of it
  - Can only send updates every second or so
  - Dead reckoning doesn't look realistic at such low update rates
- Our leverage: players not very interested in these objects
  - Just needs to look realistic, doesn't need to be exactly right
  - If not focused on an object, only will notice if it's acting oddly
- Our approach: Guidable AI
  - Players not sending frequent updates instead send predictions of where they'll likely be at time of next update
  - Those players get simulated locally by AIs, which follow natural-looking paths to accommodate predicted behavior
  - Prediction also encompasses behavior, such as how "shooty" and "jumpy" the AI should act
  - Implementation can leverage existing AI code written for bots

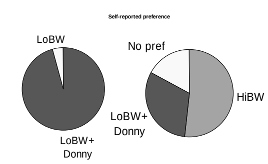
Evaluation

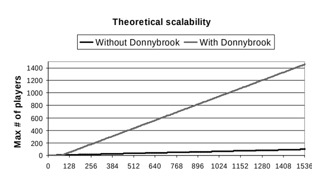
- Implemented Donnybrook techniques on P2P Quake III
- Evaluated "fun" with user study
- Compared three versions
  - **LowBW**: Current state-of-the-art in low-bandwidth setting
  - **LowBW-Donny**: Donnybrook in low-bandwidth setting
  - **HighBW**: Quake III in LAN setting











- Conclusions
  - Donnybrook significantly better than state-of-the-art bandwidth compensation techniques
  - Donnybrook makes low-bandwidth game almost as much fun as if bandwidth were unconstrained
  - Donnybrook techniques dramatically increase scalability

Form your own opinion by playing our demo!

Future work

- Refine techniques for handling missiles
- Develop latency-sensitive, bandwidth-sharing multicast framework to leverage heterogeneous bandwidth capacities