# Peer-to-Peer Support for Massively Multiplayer Games

**INFOCOM 2004**

謝志峰

**2004/2/24**

# Outline

- Introduction
- Pastry
- General Distributed Game DesignDistributed
- Game On A P2P Overlay
- Experiment
- Conclusions

# Introduction

- We present an approach to support massively multi-player games (MMGs) on peer-to-peer overlays.
- Games are different from previous P2P applications that focus on the harnessing of idle storage and network bandwidth, including
  - storage systems,
  - content distribution
  - instant messaging .
- We have designed scalable mechanisms to distribute the game state to the participating players
  - to maintain consistency in the face of node failures.
  - dynamically scales with the number of online players.
  - more flexible
  - lower deployment cost than centralized games servers.

# Introduction

- Three potential problems must be addressed to make this approach fully applicable in practice:
  - **Performance**—
    - games have frequent updates
    - under certain time constraints.
    - peers have limited bandwidth
  - **Availability**—replicating game states to improve availability has two potential problems.
    - once offline, its state becomes stale and the replica becomes invalid.
    - high update frequency, maintaining a large set of replicas is a potential performance bottleneck.
  - **Security**—
    - the prevention of cheating during game
    - Distributing game increases the opportunities for cheating.

## Pastry

- Pastry maps both the participating nodes and the application objects to random
  - uniformly distributed IDs from a circular 128-bit name space
  - implements a distributed hash table to support object insertion and lookup

- Objects are mapped on the live nodes whose ID is numerically closest to the object ID.
  - we have four nodes with IDs 1, 3, 7 and 10, then message 4 will be routed to node 3, and message 8 to node 7.

## General Distributed Game Design

- Our idea is to distribute the transient game state of the MMGs on a peer-to-peer network.

- The system could also be used
  - without a server for ad hoc game sessions when hundreds or thousands of players gather together.

  - that players in games have limited movement speed and sensing capabilities, thus the data access in games exhibits both temporal and spatial localities.

## General Distributed Game Design

- We split game state management into the classes presented below.
- *1) Player state:*
  - Each player updates his own location as he moves around.
  - Player-player interactions, such as fighting and trading, only affect the states, e.g., life points, of the players involved.
  - Because position change is the most common event in a game, the position of each player is multicast at a fixed interval to all other players in the same region.

# General Distributed Game Design

- 2) Object State:
  - We use a coordinator-based mechanism to keep shared objects consistent. Each object is assigned a coordinator, to which all updates are sent.
  - The coordinator both resolves conflicting updates, and is a repository for the current value of the object.
  - Successful updates are multicast to the region to keep each player’s local copy fresh.

## General Distributed Game Design

- *3) The Map:*
  - Graphic elements for the terrain and players are typically installed as part of the game client software, and can be updated using the normal software update mechanisms.
  - Maps are considered read-only because they remain unchanged during the game play.

# Distributed Game On A P2P Overlay

- We base our discussions on Pastry.
  - we group players and objects by regions
  - distribute the game regions onto different peers by mapping them to the Pastry key space.
  - each region is assigned an ID, computed by hashing the region’s textual name
  - a live node whose ID is the closest to the region ID serves as the coordinator for the region.

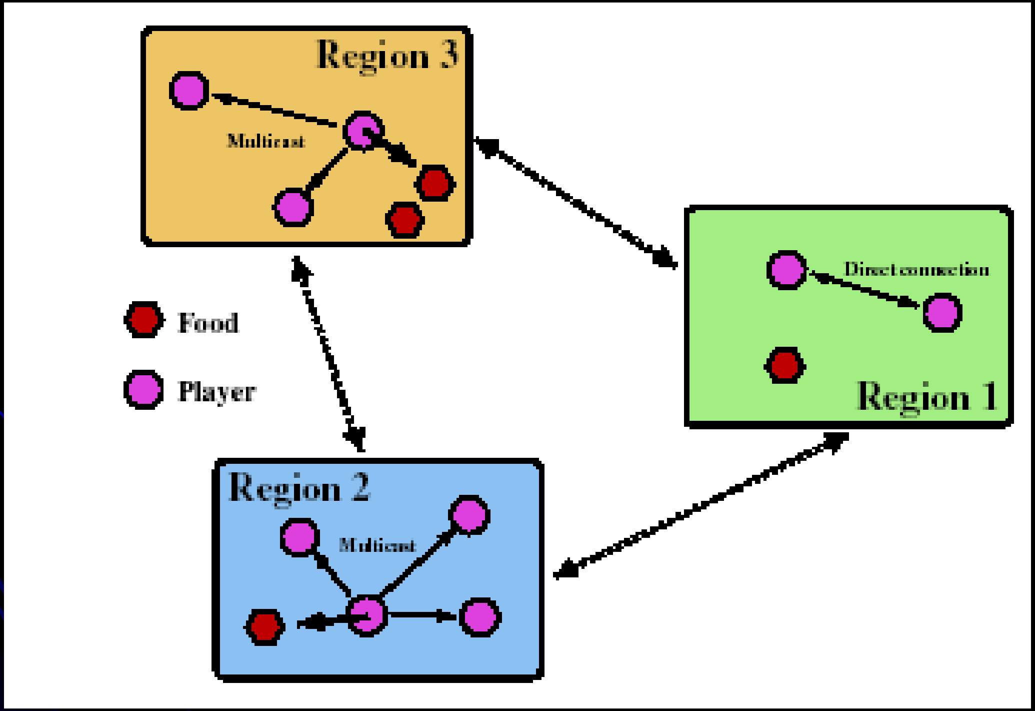
## Distributed Game On A P2P Overlay

- We design a lightweight primary-backup mechanism to tolerate fail-stop failures of the network and nodes.
  - keep at least one replica up under all circumstances, to prevent losses.
  - given an object with key *K*, then the numerically closest node *N* will be its coordinator.
  - make the next numerically closest node *M* the object replica. **($N,M : \forall I : |N - K| \leq |M - K| \leq |I - K|$.)**

- Pastry message with key *K* will always be routed to the corresponding coordinator *N*, and should *N* fail, this message will instead be routed to the replica *M*.

## Distributed Game On A P2P Overlay

- Networked games and distributed real-time simulations have exploited this property and applied interest management to game state.
  - partition the world into regions based on the limited sensing capabilities
  - players in the same region form an interest group for that portion of the map
  - state updates relevant to that part are disseminated only within the group.



## Distributed Game On A P2P Overlay

- The fault-tolerance problem:
  - The replicas must be kept consistent upon node and network failures.
  - Pastry provides limited fault-tolerance in that their routing, but game states still need to be replicated to improve their availability.

- Node failures are independent:
  - The node ID assignment in P2P networks is quasi-random.
  - There is no correlation between the node ID.

## Distributed Game On A P2P Overlay

- The failure frequency is relatively low:
  - We expect players to be online for extended periods of time and have incentives not to disconnect except when they exit (gracefully).
  - It also means we need fewer replicas of data to maintain consistency in the face of node failures.

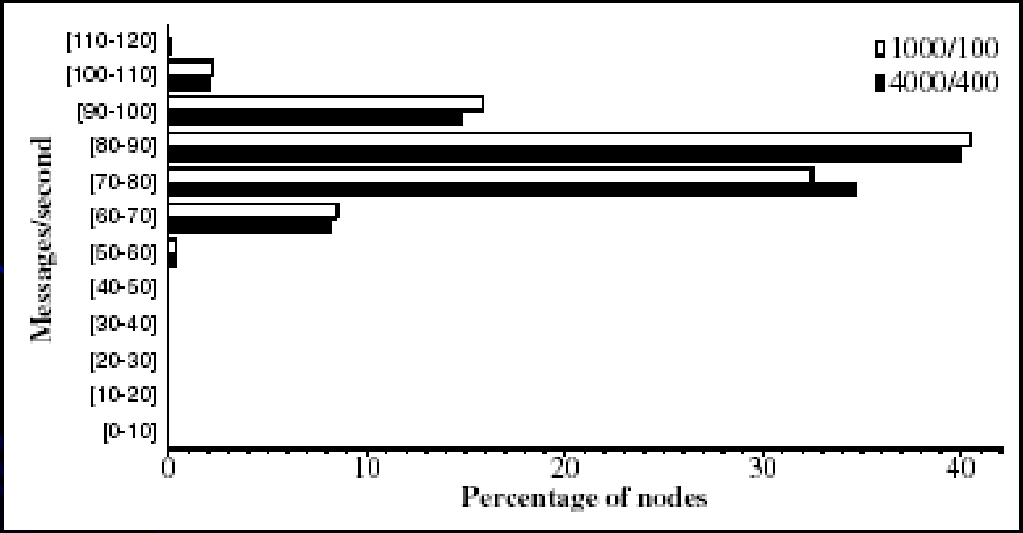
- Messages will be routed to the correct node:
  - The low failure frequency implies that a key will almost always be routed to the node whose ID is numerically closest to key.
  - With a much lower failure frequency, it is reasonable to assume that messages eventually reach the correct node.

## Experiment

- Players can perform three different actions—moving, eating and fighting.
- Each multicast message for position updates includes player ID, the current location on the map, and a player specific sequence number.
- We distribute the computational and communication load by mapping regions to the Pastry key space.
- Each region consists of a 200x300 map grid and is described using a 60 KB array.
- Associated with each region is also a 60KB object array.

# Experiment

- It presents the distribution of message rates for 1000 and 4000 players with 100 and 400 regions.
- Each node receives between **50** and **120** messages per second, which matches our expectations given the region density (10 players/region) and update frequency (about 7/second) yielding 10*7 update messages per second.
- Eating and fighting take place at intervals of 20 seconds, region change at intervals of 40 seconds.



# Experiment

- Most unicast and multicast messages are delivered within six

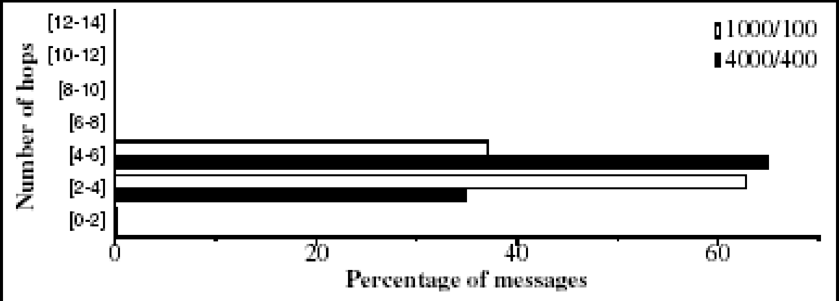


Fig. 3. Distributions of unicast message hops. Average group size is 10. No message aggregation.

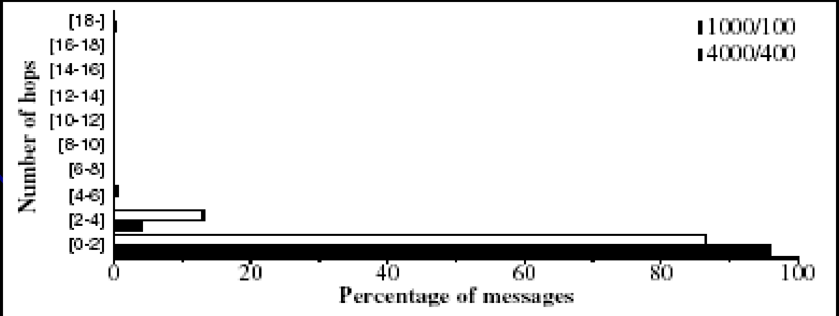


Fig. 4. Distributions of multicast message hops. Average group size is 10. No message aggregation.

## Experiment

- Since updates are multicast, this allows us to aggregate messages in the root before relaying them.
- This allows us to both reduce the number of messages and the average per-node load.

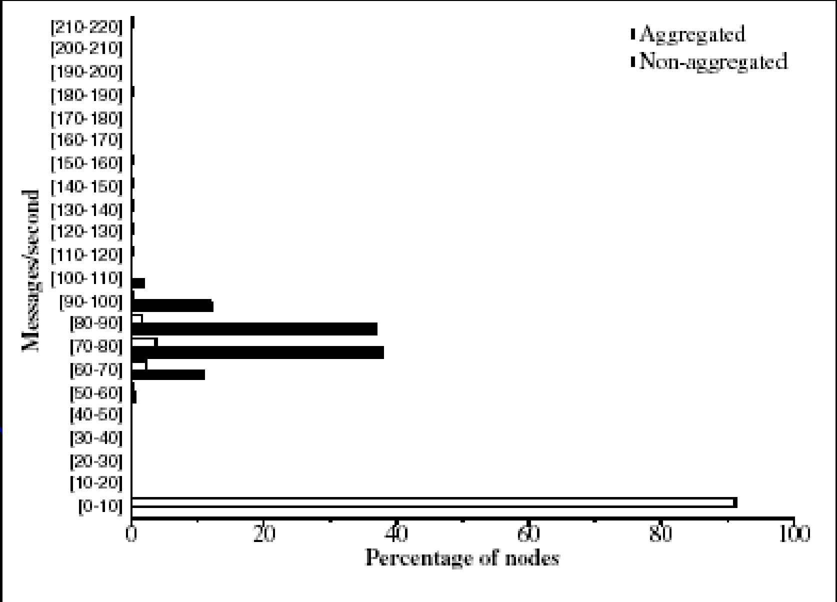


Fig. 6. Effect of message aggregations. 1000 players, 100 regions.

## Conclusions

- They have demonstrated that a new application, massively multiplayer games, can be supported on peer-to-peer overlays.
- The shared state distribution and replication mechanism presented in this paper not only can handle games, but can also be extended to handle other forms of peer-to-peer computing.
- The P2P architecture is potentially suitable for cheat detection, because locality of interest and the basic replication scheme apply to both the game and monitoring of game states.