Quiver: a Middleware for Distributed Gaming
Giuseppe Reina, Ernst Biersack, Christophe Diot
Introduction

Online Gaming is a huge business

- $16 Billion in 2010, expected $29 billion in 2016 [industrygamers.com 2011]
- World of Warcraft
  - 10 Millions subscribers
  - $15 monthly subscription

Online games rely on centralized architectures

- One or more servers manage the game simulation
- Players connect using clients
- Some Drawbacks
  - Cost
  - Robustness
  - Scalability
- First-Person Shooter games are the most affected
Introduction

Could peer-to-peer architectures solve the problem?
- Game management
  - Less load on the server
  - More complexity

Developing a peer-to-peer version of the game is a tough job!
- Game engines are extremely complex systems
  - They work thanks to the collaboration of several independent subsystems (i.e. physics and collisions engine, AI engine, Rendering engine, etc...)
  - Such autonomous systems can work only if the simulation is done locally.
- To port a game to peer-to-peer means re-engineering of the whole game engine
Our Research Question

Is it possible to create a peer-to-peer solution that can be integrated into *any* game with *minor* modifications to their engines?

Principles

- **Non-intrusiveness**
  - Few modifications to the game engine are required
- **Portability**
  - Not specific to any game engine
How does a generic game engine work?

Game state
• A collection of Entities

World Simulation
• A single threaded loop

Game state:
- A collection of Entities

World Simulation:
- A single threaded loop

- **Entity**
- **Entity modified**
- **Entity created**
- **Entity destroyed**

**Player Command Fetching**
- Command received «Shoot to the bomb»

**Player Command Execution**
- «Shoot to the bomb» executed!
Our approach

How do we address peer-to-peer gaming?

- Reuse of basic game engine designs
- Abstraction from the game engine itself through a middleware: Quiver
- Provisioning of interfaces for both the distributed algorithm designer and the game developer
- The game engine is tricked into thinking it is the only engine
The three challenges of Peer-to-peer gaming

State partitioning
- The game state (i.e. the entities) must be stored on the multiple nodes
- Therefore, each node has responsibilities for a subset of the game state

State discovery
- Dynamically fetch part of the game state that have become relevant to the node (i.e. its interests)

Simulation coordination
- Nodes can concurrently modify the game state, still they must share a consistent view of the game
Quiver

- Quiver has 3 separated modules that address the previous three issues
- Responsibility(ies) are used for distributing entities
- Interest(s) are used to discover entities
- Responsibilities are negotiated and Interests are submitted in a generic overlay network
- Copies of the same entity on different peers are kept *eventually* consistent in an application-specific manner
Quiver integration

- Entity/Memento wrapping
  - Provide the shared data-structure for both the game engine and the middleware
  - Memento keeps track of the modifications made by the game engine to the inner entity

```
| health | ammo | ...
|--------|------|------
```

```
Memento

| health | ammo | ...
|--------|------|------
```

```
Entity

<table>
<thead>
<tr>
<th>Interest</th>
<th>isModified</th>
</tr>
</thead>
</table>
```

```
Game engine

| health | ammo | ...
|--------|------|------|
```

```
Quiver

<table>
<thead>
<tr>
<th>Quiver call</th>
</tr>
</thead>
</table>
```

```
Player Command Fetching

<table>
<thead>
<tr>
<th>Graphics and Audio Rendering</th>
</tr>
</thead>
</table>
```

```
Player Command Execution

<table>
<thead>
<tr>
<th>Entity Simulation</th>
</tr>
</thead>
</table>
```
Quake II case study

Java Implementation using Aspect Oriented Programming
- Quake II source code is not modified by hand
- Quiver is “injected” using AspectJ into Quake II

Only the Server component is integrated
- Distributed Server architecture and P2P architecture

Specific solutions for Quake II
- Simulation Coordination -> Single primary replication [Bharambe 06]
- State Partitioning and Discovery -> Spatial Responsibilities and Interests
- Overlay -> DHT based on the BSP tree of the map
Simple case scenario

Game map is partitioned using disjoint Spatial Responsibilities
- Responsibilities are assigned to peers
- Entities are distributed according to the spatial responsibilities

Entities have a Spatial Interest
- The interest specifies the entities that will be accessed in the future
- Interests of the entities in a peer responsibility defines the part of the game state that should be locally replicated
Conclusions

Is it possible to create a peer-to-peer solution that can be integrated into any game with minor modifications to their engines?

Our answer...

- We designed Quiver, a middleware that abstracts from the game engine and encapsulates peer-to-peer solutions for gaming into its modules.
- Quiver requires few entry points for the integration with an existing game engine (i.e. entity wrapping and game loop).

However, we found some problems w.r.t. consistency.

Future work:

- Performance evaluation of Quiver and Quake II w.r.t. the centralized version.
- State partitioning algorithm tailored on highly dynamic environments (i.e. FPS games) with support for churn and failure tolerance.
- Distributed latency compensation algorithms.
Thanks for your attention
Questions?
Backup Slides
Consistency Limitations

Absence of latency compensation mechanisms that work in peer-to-peer
- Remote managed entities appear moving jerkily
- Common problem in centralized games
- More challenging to implement in a distributed manner

Per-entity eventual consistency is not enough
- Lose semantics of an entity modification
- Effect
  - An event (simulation or command) that access a group of entities might violate consistency constraints when one or more entities are in conflict with other remote modifications
- Need for a transactional consistency model as well
  - Must identify the semantics of the block of code that accessed the entities
  - Breaks the *non-intrusiveness* requirement of the middleware
Some scenarios

Pure peer-to-peer

Multiple servers
MMORPG Subscriptions trend
Challenges

Game distribution
- Simulation coordination
  - All players must share a consistent view of the game
  - It allows players to concurrently modify the game state and to synchronize on new version of the game state
- State partition
  - The game state (entities) must be stored on the multiple nodes
  - Each node stores the essential fraction of the game state
- State discovery
  - Dynamically fetch part of the game state that have become relevant to the player

Peer churn and failure tolerance
- The system should allow players to dynamically join and leave the game at any moment

Cheating prevention
- A malicious player should not be able to take advantage of the portion of the game state he is responsible for

Load balancing
- Diverse available resources at the peer nodes must be optimally allocated