A Publish/Subscribe Network Engine Testbed for Multiplayer Games

César Cañas
McGill University
ccanas2@mail.mcgill.ca

Kaiwen Zhang
University of Toronto
kzhang@cs.toronto.edu

Bettina Kemme
McGill University
kemme@cs.mcgill.ca

Jörg Kienzle
McGill University
joerg.kienzle@mcgill.ca

Hans-Arno Jacobsen
University of Toronto
jacobsen@eecg.toronto.edu

ABSTRACT

Massively multiplayer online games have to handle huge amounts of load caused by thousands of concurrent players and require a scalable, low-latency middleware system for their messaging needs. In this demonstration we will showcase the three pub/sub network designs that we developed for our Middleware 2014 paper titled "Publish/Subscribe Network Designs for Multiplayer Games". The demonstration will use the MMOG prototype Mammoth and the PADRES distributed pub/sub system to compare the performance and features of the network engines in real-time.

Categories and Subject Descriptors

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

1. INTRODUCTION

In the last decade Massively multiplayer online games (MMOGs), in which thousands of players share the same virtual-playground, have become extremely popular. Normally, almost all MMOGs follow a centralized client/server architecture where the server maintains the game state and the clients hold replicas of the objects in which they are interested. The server is in charge of performing *interest management*, which determines what objects are interesting for each client. The server is also in charge of transferring replicas to the clients which require them, and of updating the replicas when the state of the game changes.

Perhaps the biggest technical challenge of MMOGs is scalability: to allow as many players as possible to share the same virtual world and interact among themselves. In order to address this issue, we proposed in our Middleware 2014 paper "Publish/Subscribe Network Designs for Multiplayer Games" [1] a trio of pub/sub-based network engines that support the messaging requirements of a typical MMOG (or other location-based systems). By taking advantage of the nature of pub/sub systems, we are able to offload responsibilities typically handled by game servers, most notably, interest management to the pub/sub middleware itself. In this demonstration, we will present this trio of network engines and compare their performance and features in real-time.

Middleware '14: Demos and Poster, December 08 - 12 2014, Bordeaux, France
ACM 978-1-4503-3220-0/14/12.
http://dx.doi.org/10.1145/2678508.2678518.

2. NETWORK ENGINES OVERVIEW

We have developed three different network engines. The first two use a topic-based pub/sub system, while the last one uses a content-based approach. To be able to handle a game's messaging requirements, the network engines must perform the following basic operations: a) allow clients to discover new in-game objects as they move, b) transfer replicas to the clients that require them, and c) propagate any state update from a master object to all of its replicas.

All network engines assign a topic to every client node and every in-game object in the system. Clients subscribe to their own topics and can be contacted by publishing a message on that topic. Likewise, nodes can subscribe to an object to receive updates about it.

Our game worlds are divided into smaller sections known as tiles. These tiles are triangular in shape and obstacle-aware: they never cross impassible boundaries like in-game walls. Some of our network engines take advantage of the properties of these tiles to improve their performance [2].

2.0.1 Network Engines

- Object-Based Network Engine: This engine, unlike the others, requires at least one node to run an interest management service. This service keeps track of player/object locations and determines when a client becomes interested in an object, at which point the service makes sure that a replica of the object is published under that client's topic. After receiving a replica, the client subscribes to the corresponding object. In this way, to propagate a change on an object's state, the update information is simply published under that object's topic, to be received by all interested suscriptors.

- Tile-Based Network Engine: This engine assigns three special topic channels to every single tile in the game-world, as shown in Figure 1. These channels are used for most communications between nodes. The nodes that contain master copies of objects, known as *master holders*, subscribe to the *replica request* channel of the

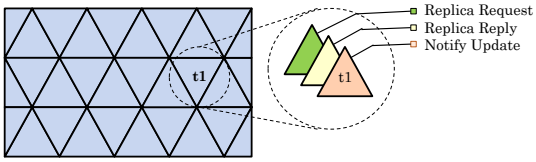

Figure 1: Tile-based network engine channels

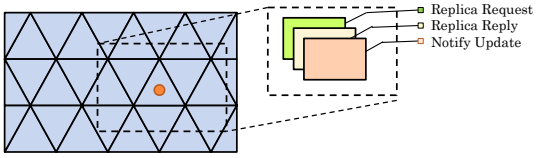

Figure 2: Area-based network engine channels

tiles where those objects are located. Client nodes subscribe to the *notify update* channel of all tiles they are interested in. When a player moves around the map, it determines by itself which tiles have become interesting and uses the *request replica* channel of those tiles to ask the master for any replicas located in the tiles. The master holders will then send the replicas to the clients using the *replica reply* channels of those same tiles. Propagating an object's state change is done by publishing the update in the *notify update* channel of the tile that contains that object.

- Area-Based Network Engine: This engine is similar to the tile-based one. It also uses three special topics but, instead of subscribing to the in-game tiles, nodes use a content-based approach to subscribe to rectangular areas of the map, as shown in Figure 2. The master holder of an object subscribes to an AoI rectangle on the *replica request* topic that is centered on the object's position. A client, on the other hand, subscribes to the *notify update* channel of an area centered over the player they control. When the player moves around it publishes a petition for replicas in the *request replica* channel, centered over its player. The pertinent master holders will receive this request and send the appropriate replicas back, using the *replica reply* channel, making each publication on the position of the objects. Propagating an object's state change is done by publishing the changes on the *notify update* topic, using the current position of the object.

3. DEMONSTRATION

Our demo will showcase and compare our three network engines. First, we will present the Mammoth game engine, explain our system setup, and describe to the public the performance measurements that we will keep track of.

Our system setup consists of a couple of laptop computers running a Mammoth player client similar to the one shown in Figure 3. Conference participants will be able to control the actions of a few player characters, move them around the map, and use the command console to give orders to npc-controlled characters (to make them follow a player's character, for instance). The npc client nodes, brokers, servers, and other components of the system will run on a cluster of

Figure 3: The Mammoth MMOG main screen

Figure 4: Pub/sub messages comparison graph

computers located at McGill university to which the local client machines will connect to. The components of the system will save the performance metrics, such as the number of subscriptions or the network load, on a database which will be queried by another laptop computer in charge of displaying real-time comparison graphs and performance metrics for each network engine.

In order to properly compare the network engines, during the presentation our system will run all three of them simultaneously. One of the network engines will be fully operational and handle all network communication, as required by the game, while the other two will work under a "dummy mode" where they perform their usual tasks and save their performance metrics in the database but do not actually publish any messages. By doing this it will be possible for the participants to perform an action as see the number and type of messages that each network engine generates in response to it. An example of the type of comparisons that will be produced is shown in Figure 4

We will finish the demo with our observations about the advantages and disadvantages of each network engine.

4. REFERENCES

[1] C. Canas, K. Zhang, J. Kienzle, B. Kemme, and A. Jacobsen. Publish/subscribe network designs for multiplayer games. In *Middleware 2014 - ACM/IFIP/USENIX 15th International Middleware Conference, Bordeaux, France, December 8-12, 2014, Proceedings*, 2014.

[2] J. Chen, B. Wu, M. DeLap, B. Knutsson, H. Lu, and C. Amza. Locality aware dynamic load management for massively multiplayer games. In *POPP*, 2005.