ECE419 Lab 2 Design Draft

Jay Suthar (995474749) and James Robinson (996185809)

Feb 12, 2010

1 Class Techniques Used

1.1 Logical Vector Timestamps

Every packet sent by a node will be sent with a vector timestamp, containing the Lamport clock of each node currently playing in the game. On every event (joining, moving, firing, receiving, sending, dying) the current node's timestamp will be incremented, allowing all events to be arranged in a logical sequence. Nodes will be assigned an order of precedence at a place in the timestamp data structure during the initial startup communication.

We chose this method because it is needed to implement synchronization among the many independent nodes, and considerably simpler than a simulation of a physical clock. It isn't particularly scalable, since the size of the timestamp grows with each client, as well as the initial set-up time to configure the data structure, but with a maximum of 4 clients, we don't anticipate any particular problems.

1.2 Totally Ordered Multicasting using Out of Order processing

We will be using those logical vector timestamps to order a queue, implemented on every node, to ensure consistency. A node will only update its state after it knows that every other node has the same event in its queue (as symbolized by receiving an ACK from all other clients).

We chose this method because it is conceptually the simplest method we know to ensure consistency among all nodes using logical time stamps, and ensuring complete consistency among nodes is a hard constraint of this lab.

1.3 Timeout-based failure handling

Failure handling will be implemented using the concept of a timeout. If a node does not ACK within a certain time, it will be dropped from the game, and the other nodes will proceed without it, not requiring its ACK to proceed with their queues. This ensures that catastrophic failure for one client does not destroy the game for everyone else.

We chose this method because we don't know of another way of detecting fail-stop failures, and we want to get the 15% bonus. :)

2 Other planned techniques

2.1 jSLP

We plan on using jSLP (`http://jslp.sourceforge.net/`) to implement initial service discovery. This allows automatic service discovery, removing any need for long command-line lists of node hostnames and ports, making the game appear much more user-friendly.

We chose jSLP because it's a pure Java open-source implementation of SLP, which seems designed for exactly what we're trying to do: automatic service discovery. We realize it'll be more work, but hopefully, the jSLP library will decrease the amount of effort required, and will result in a significantly slicker network initialization.

3 Expected Scalability and Efficiency

Our scheme is not particularly scalable. Due to our broadcast-based networking protocol, network traffic will increase exponentially as more nodes are added, making this completely impractical as the number of nodes approaches infinity. On the other hand, the fact that our solution is completely distributed does mean that there is no single point of failure, meaning failures are not a bound on scalability. Due to the fact each node now plays the roles of client and server though, this only makes the implementation less scalable in terms of CPU time than the client-server solution. Of course, the client-server solution is only worth a maximum of 40%, so it wasn't even an option. :)

The vector timestamps used for synchronization are also not particularly scalable, increasing linearly in size as the number of nodes increases, which will also increase network traffic. However, considering the small size of integer values, the short lengths of games, and the 4 node cap, we're again not particularly worried about scalability in practice.

SLP, the protocol used for service discovery, has evidently "been designed to scale from small, unmanaged networks to large enterprise networks."[1], so it actually might be the most scalable part of our design. On the other hand, using SLP will be considerably less efficient than simply connecting directly to different computers, and therefore, we cannot classify our design as efficient either.

Due to the cap of 4 nodes total, and the fact our game will be running on a LAN, we're not particularly worried about network scalability. Also, considering the computers it will be running on are all quite new, we don't think CPU latency will be particularly high either. As a result, despite the fact our game isn't designed to be scalable or efficient, we don't expect the game to play slowly. If it does, we'll profile the code and figure out what went wrong, but we don't foresee any particular difficulties in that regard, due to the superior hardware it'll be running on.

[1]`http://en.wikipedia.org/wiki/Service_Location_Protocol`