ECE419 - Lab 3 Design Report

Kevin Gumba 997585117
Tahia Khan 998897216

Monday March 24, 2014

1 Overview

This peer-to-peer implementation of Mazewar supports dynamic joining for four players. It employs the use of Lamport clocks and the Ricart and Agrawala algorithm to enforce synchronization and mutual exclusion. Its main components include a Lookup/Naming service and a multicasting system for client communication.

2 Components

2.1 Lookup/Naming Service

A lookup service is used to coordinate joining and exiting of clients. It does so by storing all clients within a concurrent hashmap data structure using a key and value. The key signifies client ID and the value is a class named ClientData that holds all the information the client has sent to the lookup service (ex. hostname, port). It is assumed that the lookup service is always running.

When new clients request to join the game, they are asked for their game name, port they will be listening into (Refer to Section 2.3 for client Listener), as well as the current hostname and port of the lookup service. Clients then connect to the lookup.

If clients choose a port that is currently in use by another client, the lookup sends a disacknowledgment and the client must try a different port. Once the lookup service acknowledges the client registration, a client ID is given to easily identify and organize clients. In addition, the lookup also sends a copy of the concurrent hashmap lookup table containing the other clients' port and hostnames.

When a client requests to quit the game, it notifies the lookup service which removes the client from the table.

2.2 Clients

Each client consists of a ClientHandler, a Listener and a Dispatcher. The ClientHandler is responsible for organizing the event queue, running event packets to update the client screen, and communicate with the lookup service. Each event packet shall include the client's ID, Lamport clock timestamp, type of event (ex. left, quit, etc), event information (ex. point, direction, score, etc). The Listener listens for packets on a chosen port from peers and adds relevant incoming events to the event queue. The Dispatcher handles all outgoing packets of the client, including key press actions, acknowledgments, etc.

2.3 Communication Protocol

When a client wants an event to occur, it will first broadcast a proposed Lamport clock timestamp for that event. The requester then tries to acquire semaphore of size N-1. Other clients will verify that the proposed clock is usable and send an ACK if the clock is up to date and increment their current lamport clock. If a NACK is sent, the updated lamport clock timestamp shall be included in the packet. When the requester receives an ACK or NACK, a semaphore is released until it reaches N-1. If the proposed lamport clock timestamp has been disacknowledged, the requester must retry with an updated lamport clock timestamp.

After broadcasting and receiving acknowledgements from other clients that the proposed Lamport clock is valid and can be used, the client shall broadcast the packet containing all information on the event to all clients including itself.

When a client receives such event message, it sorts the messages by lamport clock and adds it to its local FIFO event queue. When the event is extracted from the queue, it updates its game screen.

3 Game Details

The following is a description of the general game logic, starting from how a client join is handled, how communication is maintained throughout the game, and client quit.

- Upon start-up, a new client registers itself to the lookup service and receives an ID and a copy of the lookup table containing information (e.g. hostname and port) of all existing clients. The new client connects to all other clients using the information in this table.
- The client dynamically joins the game by broadcasting itself to all other clients. Existing clients acknowledge the request by responding with the current location/direction of their avatar, so that the new client may spawn at an appropriate location. The newly spawned location is broadcast to the other clients.
- Handling events
    - Broadcasting own events: on key presses, client broadcasts its current lamport clock timestamp (ie. to maintain a global clock)
    - All other clients update their local lamport to 1+max and sends ACK or attaches their more up to date timestamp and sends a NACK
    - Requesting client blocks with semaphores until all acknowledgments to the lamport timestamp are received
    - Once acknowledgement received, client creates an event packet with the lamport clock and broadcasts the packet. It also adds the event it created to its event event queue and increments the lamport clock timestamp
    - All other clients receive the event packet and adds it to the event queue
    - Event packets tagged with the corresponding timestamps are extracted from the event queue and are sent to the console to update the client screen
- When quitting, client requests the lookup service to remove itself from the table and broadcasts a quit event to all peers

4 Design Evaluation

4.1 Starting, Maintaining, and Exiting

The startup of the game relies on the lookup service. The strength of the lookup service is that it is a light-weight and centralized component of the system which provides a simple way to keep track of players and register a new client into the game. The new client sends one message to the lookup for registration and the lookup replies back with an acknowledgment, totalling two messages. The drawback is that since it is centralized, the Mazewar game relies on the proper functioning of the lookup- if it crashes, all player information will be lost and the game becomes corrupted. This could be handled by saving this information to disk after some time interval. The lookup also acts as a bottleneck since all clients must use the lookup service for all registration and exit events.

Maintenance of the game is robust because of the use of a global Lamport clock alongside the Ricart and Agrawala algorithm. Since any Lamport timestamp associated with an event packet must be

acknowledged and approved by all peers before it can be issued, total ordering is achieved in the game. The fallback of this algorithm is the large number of messages that must be issued for any single event: first, a requesting client broadcasts a message to all (N-1) peers to request a Lamport clock; then, the client waits to receive (N-1) acknowledgements from all peers; lastly after all ACKs are received, the actual event packet is broadcasted to N client. Since the Ricart and Agrawala algorithm is used, a reliable network connection is necessary since failure handing, packet drops, and corrupt data transfers are not handled for.

Exiting in the game consists of broadcasting a quit event (N messages) and removal of the quitting client from the main table in the lookup service (1 message). Similar to the drawbacks stated above, the lookup service still acts as a bottleneck and the network connection is required to be reliable.

4.2 Performance

Timestamp are placed before a packet is sent out and after a packet has been broadcast and placed into the local queue. The latency of each packet has been calculate, with fastest packet delivery with only one client (25 ms / packet) and slowest with the maximum of four clients (32 ms / packet).

# of clients	Lamport timestamps	Timestamp (ms / packet)
1	L(0) to L(14)	(1395676114092 - 1395676113792) / 15 = 25
2	L(0) to L(11)	(1395675650276 - 1395675649972) / 12 = 26
3	L(0) to L(19)	(1395676584124 - 1395676584727) / 20 = 30
4	L(0) to L(18)	(1395677068337 - 1395677067729) / 19 = 32

4.3 Scalability

The game is able to handle four players in a small LAN with no major performance issues. If it is played across a higher-latency, lower-bandwidth wireless network, the amount of data transfer shall decrease since the algorithm used relies on broadcasting for lamport clocks and event packets. Since multiple clients are broadcasting in parallel, a higher-bandwidth would be ideal in order to transfer all packets in parallel.

Finally, since fault tolerance is not handled and the algorithm requires a reliable network connection, packet loss rates, would lead to problems such as lost wakeups, lost events, and inconsistent screens.

4.4 Consistency

Since the algorithm requires a reliable network connection, large time delays in data transfers can lead to inconsistencies. For example, if an event queue is waiting to extract the next event packet with timestamp 8, yet all others are recieved, a lag shall appear on that clients screen. Other packets can accumulate such as packet 9 to 15. Once the event timestamped 8 is received, event packets 8 to 15 are run and can appear as a speedup of the game on the client's screen. In addition, packet reordering over the network can cause unusual gameplay. For example, if a client presses left and right away right, if the packet sent out is correctly but reordered over the network, unwanted movement would occur.