CS244b Assignment 1:

Mazewar Answers

Alhaad Gokhale <alhaad@stanfrod.edu>

April 21, 2016

1. **Evaluate the portion of your design that deals with starting, maintaining, and exiting a game - what are its strengths and weaknesses?**

I think that the portion of our design that deals with starting, maintaining and exiting a game has the following strengths and weaknesses:

Strengths**:**

- **Our design is fairly easy to implement**. We designed our protocol with implementation simplicity in mind. We did not bake in any complicated handshake mechanisms, order or delivery guarantees. This means that our protocol is well suited for its purpose - to be implemented as a part of a class and run in a controlled environment. A more complicated protocol could have served a more specific purpose / use-case which I discuss in the weakness section and as answers to other questions.
- **Joining and leaving a game is trivial.** Joining the game just means observing all the packets for some time and leaving the game just means stop sending packets.
- **Game state converges eventually.** Our protocol is resilient to packet drop, partitions in the network or a slow / non-responsive node in the game. Eventually (i.e. in steady state), all players would agree on score, no two rats will be in the same square and projectile hits at most one person. More on this in the answer to question 3.
- **Fairness.** Our protocol makes use of a frequently generated random token to resolve conflicts if two rats try to enter the same square. This has better fairness properties than say always preferring the rat with a lower ip address.

Weaknesses:

- **Temporary inconsistencies**. In some pathological scenarios there might be temporary inconsistencies like if a rat has left the game, other players still see it for a small amount of time. In such a scenario, a missile might pass through it but our protocol ensures that the shooter does not get 2 hits from the same missile.
- **Possible player id collision**. Our protocol makes use of a randomly generated 32 bit id for each player and does not resolve conflicts. Though the possibility of 2 32 bit IDs colliding in a small LAN environment is low, if it were to happen the result is undefined.

2. **Evaluate your design with respect to its performance on its current platform (i.e. a small LAN linked by ethernet). How does it scale for an increased number of**

players? What if it is played across a WAN? Or if played on a network with different capacities?

Our protocol is well designed for the objective use case (i.e. a small LAN linked by ethernet) and this claim can be substantiated by the fact that there are 2 implementation that work well in the provided environment.

How does it scale for an increased number of players? Our solution which relies on multicast messages does not scale well. The very fact that there are nC2 messages passed makes such a design inherently not scalable. A solution which is scalable should only have at-most n messages.

A possible design that makes use of finger tables (see https://en.wikipedia.org/wiki/Chord_(peer-to-peer)#Finger_table) would alleviate the scalability problem but might introduce other issues (difficulty of implementation being the most important).

What if it is played across a WAN? WAN, which means added latencies would be handled by our protocol fairly by simply tweaking some timeouts.

A better way of handling higher latencies would be to reduce the amount of messages we send using optimizations like piggybacking, predicting path of missiles, etc.

Or if played on a network with different capacities? Assuming that this means higher packet loss, our protocol would handle it well since we rely on passing the whole state (NOTE that the whole state fits inside a single UDP packet) as a heartbeat. Though this means that there may be temporary inconsistencies, we will get eventual consistency.

This is because sending the whole state makes the probability of packet loss multiplicative. An illustration:

Probability of packet loss = 40% (say)

Heartbeat interval = 250 ms

Probability that the state does not become consistent in 1 second is 0.4*0.4*0.4*0.4 = 0.0256

The probability that the state will become consistent in 1 second is 97.4%

3. **Evaluate your design for consistency. What local or global inconsistencies can occur? How are they dealt with?**

Our protocol is well designed to maintain consistency and guarantees eventual consistency. What 'eventual consistency' means is that though there might be temporary inconsistencies, these consistencies will resolve themselves given sufficient time. Following are some examples of local and global inconsistencies:

(1) If rat A, moves from square 1 to square 2 to square 3, it has to send a new packet every time which might get dropped or reordered. This means that rat B might see rat A jump from square 1 to square 3 without going to square 2. This can be dealt with by baking explicit acknowledgements (via 3 way handshake) for every message sent.

(2) Another problem similar to the previous bullet is missiles jumping squares which would look as if the missile passed right through a rat. This can also be fixed via explicit acknowledgement for every packet.

(3) Our protocol prevents two rats being in the same square by using a random token for collision resolution. There could be a scenario where rats A and B try to get into the same square via a resolution token but miss the packet which contains the random

token. This means that rat A and B believe that they are not in the same square but a rat C looking at them and which has received all packets might see bot of them in the same square until one of them moves out. This would require these rats to come into some kind of global consensus.

4. **Evaluate your design for security. What happens if there are malicious users?**

Our protocol was designed without any security measures in place and trusting all participating nodes.

Assuming that there are no malicious nodes. Since communication happens over a multicast channel via plain text and with no identity verification in place. A third party node could observe the communications, spoof by acting on behalf of other players, make the game state inconsistent and even DDOS all the nodes.

What happens if there are malicious users? A malicious user can trivial cheat in the game and also do what a third party user can.

TLDR-No security :)