Answers
Hong Zhang
romanzhg@stanford.edu

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

For starting the game, there is no special control packet needed, a user keep on sending the state packet, and exam the "reject" field of USERNAME-REQ packet. If there is more reject than accept, the user quit. This simulates the Quorum mechanism, so even if there is more user than the maximum in the game due to temporary network separation, the inconsistency will be resolved overtime, and extra user will be removed.

For maintain the game, we send state instead of action, this ensures inconsistency won't accumulation overtime. The join process is designed And we try to minimize the packet size, which reduce the workload of the traffic.

For exiting the game, a user is though to be exit if he send out the leave message or did not send out any packet for 10 seconds.

Our design try to minimize the kind of packets, so the program has less states. Also a major intuition of this design is that drop packet/link broken is indistinguishable from program crash/sudden exit, so we should handle it in a uniformed way.

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 design works well for the current platform. Since we use multicasting, the traffic load will grow linearly as the number of player increase, so we expect it works well with increased number of players. For players across a WAN, the packet delay will be increased significantly, so the game experience will be affected, and more temporary inconsistencies will emerge. For players in a network with smaller capacity, the packet drop rate will increase, this also increase temporary inconsistencies.

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

For example, 2 player in the same place, so both of them will move randomly to a near by position.

Other inconsistencies is covered in the answer of question 1.

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

The design has no security consideration. There is no authentication and identification implemented, and the network used is udp multicast, which has no traffic control. Also the packet sent is not encrypted. So a malicious user can:

1. Send fake packages, pretending tagged others, and get scores
2. Start a lot of instance so force other user to quit the game
3. Send enormous packets to make network congestion, so perform the DDOS like attack