CS244B Mazewar Protocol Questions and Answers

Prakash Surya <surya1@stanford.edu>

April 23, 2012

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

   The design that my group and I came up with has a number of strengths including resiliency despite packet loss and reordering. Since all of the information a client needs from a remote client is sent with each state packet, consistency remains even with a high amount of packet loss.

   The acknowledgement of "tagged" packets also keep scores consistent even if a number of these packets are lost. The client will keep retransmitting, trying to let the other client know he should be awarded the tag.

   If a leaving packet is lost, the timeout mechanism will correct this.

   Nickname packets are continuously sent out at regular intervals. This is unnecessary once a client receives a single nickname packet from a given client, but it keeps it simple as these packets need not be acknowledged.

   One issue with the design is the amount of communication sent over the network, even if there is no changes in state. If this design was to scale to millions of players, the number of packets sent over the network would need to be reduced to avoid congesting the network (which could potentially cause increased packet loss, latency, and reordering).

   Another big drawback of the protocol is the fact that it doesn't deal with GUID collisions. Thus if two clients had the same GUID, these packets would get interpreted as a single player. This is very unlikely given a decent 64-bit PRNG, but theoretically it is still possible.

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?

   Testing has proven that the protocol works very well on a small LAN linked by ethernet, or even a highly congested WiFi network. It was tested in a conference environment with 200+ people using a particular WiFi access point and despite have many packets being dropped, the game was still very playable.

I think the protocol might break down over the WAN because the latencies would grow to much larger values. Thus, it would be tough to maintain a globally consistent state between all the players without more local processing. For example, each change in position of a missile does not have to be sent over the network. Given the initial position, its future positions could be extrapolated locally on the client.

The protocol should work just fine on networks with different capacities, given that the networks are somewhat reliable. It would be difficult to play on a network where most of the packets are dropped, although any network with "good enough" successful transmission rate and latency should work.

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

   It is possible for local inconsistencies to occur with there is a severe amount of packet loss. During testing we found that missiles would sometime fly through other players as a result of dropped packets. The global state generally stays sufficiently consistent between all clients as a result of each client sending it's absolute information with each state packet. So a client just needs to receive a single recent packet to bring it's local state in sync with that remote clients state.

   Another inconsistency that can occur is GUID collisions. This is simply dealt with by using the fact that it is impractical for two random 64-bit values to collide given a sufficiently random PRNG.

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

   A malicious user could wreak havoc on the game as there are not any anti-cheating mechanisms built into the protocol. For example, a user could set hit score to any arbitrary value, and the remote clients would accept and trust this value blindly.

   One could also send out malicious "tagged" packets, causing non malicious clients to award themselves tags even when this never happened. Thus, not only can a malicious user control it's score arbitrarily, it can manipulate the other clients scores.

   Malicious CRT values can be used to win all state change collisions, or always lose these collisions. This could potentially be used to manipulate the movements of the other players.

   Another big vulnerability in the protocol is GUID spoofing. A malicious user could send packets with a spoofed GUID value. This could cause clients to think a new player has joined when this isn't the case, or worse, cause clients to think an existing player has sent packets which it hasn't.