Mazewar Protocol Discussion

Song Han songhan@stanford.edu

1 Question 1

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

Our design is robust against packet loss and even under 50% packet loss rate it's still playable.

1.1 Starting Game:

A new player can join in the game at any time when there is an empty slot (totally 8 slots) in the group. When a new player enters the game, he will first listen to the packets for 5 seconds, this makes sure that he has collected other players heart beat, in this way he could collect other player's name.

ID conflict prevention: The ID selection mechanism guarantees no ID conflicts if there's no packet loss. Unlike using a random number generator to decide a random 16-bit or 32-bit ID number, which might have conflicted ID, we are scanning the free list to find the 'first fit' position as the player's ID number, guaranteeing no ID conflict.

ID conflict resolution: when there's a packet loss causing ID conflict, say during the 5-second listening packet loss happened, my program can re-select a new ID to maintain consistency. This is achieved in the following way: the player with a smaller sequence number yields this ID by cleaning his record and re-joining the game, during which will choose a new ID.

Another smart thing is when requesting new player's name, the sender is putting its own name in the NameRequest packet, thus the receiver can read out the sender's name while write its own name to the NameReply packet. This reduced the number of name request packets.

The first weakness is the user ID is constrained to support 8 users because we are allocating 8 entries in the free list. Another weakness is when the network is partially connected, say A only connects with B, B only connects with C, A and C might have the same ID, thus if A's sequence number is smaller than C, then all then packets sent by A is ignored by B completely.

1.2 Maintaining Game:

In order to keep the score constant, we discover that simply sending and receiving score itself is not enough. The most fundamental thing that decides the score is the **hit count matrix**: since a hit count matrix uniquely decides each player's score, but a player's score can results from different hit count matrix. Thus we choose the **hit counts**, instead the score, as the communication primitive. In this matrix, element $H_{ij}(i \neq j)$ means the number of times that rat i hit j. Element H_{ii} means the number of missiles that rat i has launched till now.

$$H_{m,n} = \begin{pmatrix} H_{0,0} & \dots & H_{0,7} \\ \vdots & \vdots & \vdots \\ H_{7,0} & \dots & H_{7,7} \end{pmatrix}$$

The advantage of introducing H matrix is better consistency: H matrix is more fundamental than scores. Each player keeps its own copy of the H matrix, while sending out his row H[MY_ID][:] in the heart beat. Each players' score can calculated from the H matrix.

The weakness of maintaining the H matrix is scalability. Since the size of the matrix is N^2, it can grow very large if there are more players. And the packet size is a row of the matrix, which grows linearly with the number of players. So there's a trade off between the packet size and the degree of consistency.

1.3 Exiting Game:

When a player P leaves, every player zeros P's row in the hit count matrix. To keep the scores consistent, we also maintained a base score for each player to represent the score previous contributed by P. Base score of a new player i is zero when he joins the game group. Once a player other than i leaves, it sets the base score vector according to the hit relation between this leaving rat.

We can tell that from hit count matrix H and base score vector B, we can calculate all playing rats' scores correctly at any time:

$$score_i = \sum_{j=0, j\neq i, H_{ij}\neq -1}^{7} H_{ij} * 11 - \sum_{j=0, j\neq i, H_{ji}\neq -1}^{7} H_{ji} * 5 - H_{ii} * 1$$

The strength of exit handling is that we can detect game exit by two methods: detect by GAME_EXIT packet(sent just once, possibly get lost) and detect by EXIT_TIMEOUT. Thus even the GAME_EXIT is lost we can still remove the player after timeout.

The weakness of game exit is we are sending the GAME_EXIT only once, thus it might get lost. Therefore, we set a 10 second timeout to compensate. The reason we choose a relatively long time is to accommodate packet loss case. Under severe packet loss, we need to discriminate temporarily not receiving heart beat versus permanent exit. After experiment we choose 10 seconds as the balanced timeout value.

2 Question 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?

On current platform the performance satisfying: a new player can join the game as soon as there is a free slot; Missile tagging detection is instantaneous for both shooter and the victim; States update delay is only constrained by the heart beat interval, which is currently 500ms.

With respect to scalability, the packet size grows linearly with the number of players, since each player need to send its row of the hit count matrix; The size of shared states grows quadratically with the number of players, since each player need to main a hit count matrix. Also the multicast mechanism is inherently not scaling well, the number of packets also grows quadratically with the players. So the game does not scale well with the increased number of players, there network might be fully saturated by heart beat packets if the of players are too large.

If the game is played across the WAN, there will be more latency and higher chance of packet loss. It might happen that a shooter sends out a missile and by the time the victim gets the missile location, it already moved away from the missile's original location. Or a victim gets the missile that is no longer there any more. In a word the received heart beat packet is out dated. But although there will be some latency of score card refresh, the score card among players is still consistent.

To deal with possible packet corruption in the WAN environment, we set check-sum for each packet. Packets with incorrect check-sum will be discarded. Thus it prevents wrong information from propagating.

For different network capacity, we could use different heart beat rate: for low capacity rate, lower heart beat rate could be used. We did not include this auto adjustment mechanism, but this should be an effective way.

3 Question 3

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

For game enter: waiting for 5 seconds almost guarantees the new rat to hear from all the current players, and then decides its ID by scanning the free list. However, on severe packet loss there might still missing some rats' heartbeat, thus lead to conflict ID. This is solved by letting the rat with a smaller sequence number to re-enter the game and choose another ID.

For rat's position: lost of heartbeat packages will cause outdated rat location. In the 3D view, what we see might be an old view. When the packet transmission resumes, some rats might find himself at the same location as another rat. Then the location conflict solving routine is invoked and a nearby position (within 3x3 distance) is randomly chosen to break the conflict.

For missile tagging, reborn and score calculation: since each player sends out his row of the H matrix, which means how many missiles has he shot other people, the tagging is decided by the shooter. If the shooter did not get the update of a victim's location due to a packet loss, he might think the missile hasn't shot the victim, thus even if the victim thinks he has been killed and reborn himself, the score doesn't change for both of them. On the contrary, if the shooter has killed a victim but the heartbeat packet was lost, the victim doesn't update his H matrix. However, when he receives another heartbeat packet from the shooter later, he will updates the H matrix and thus his score, which reflects him being killed, although he will not reborn himself since he doesn't think he is killed. In a word, the local inconsistency is reborn on no score reduction and no reborn on score reduction, while there's no global inconsistency on the score card.

4 Question 4

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

In our design we added check-sum for each packet to partially prevent packet corruption. However, this can not fully prevent cheating if a malicious player modifies both a packet field and the check-sum. Since the tagging is reported by the shooter, the shooter can cheat by sending out arbitrary hit counts to the rest of the rats. While the victim rat can prevent being tagged by temporarily not sending heart beat packets. Thus he temporarily becomes 'invisible' to other rats. He can not be invisible for too long otherwise other rats might think he has quit the game.