Mazewar Protocol Specification

Song Han songhan@stanford.edu
Tai Guo taig@stanford.edu

1 Introduction

CS244b Mazewar is a distributed multiplayer game. In this game, each user control a rat which can move and launch missile to hit other rats. This is a real-time distributed system with no central control node and can support player to join or leave the game group at any time.

2 Protocol Description

Our Mazewar protocol design the way in which each terminal computes the states of the game and what and how they communicate with each other. The protocol is built on UDP multicast and deal with both the stable condition and unstable condition of the network.

Our design support all the regulation of this game. We limit the maximum number of outstanding missiles of a rat to be one.

2.1 Protocol Goal

Our protocol achieves the following goals:

1. Real time playing. This means at any time, the instance cannot stop or waiting something in order to continue playing. From the view of the player, his rat is always movable, his score should change according to the event, and the rat he hits should die at that point. However, once there is some inconsistency happening, it is tolerable to change the score, the position of the rats and so on some time later, which does not affect playing.

2. The protocol must avoid inconsistency from accumulating. In other words, at a given time t in the future, if the network works ideally well around t, the protocol should recover the system and make all the instances of the program consistent. However, if the network is bad now, some temporary inconsistency is acceptable.

3. The system should not have central failure. Others can continue playing if some nodes fails.

4. A new player can join in the game at any time when there is an empty slot (totally 8 slots) in the group. An old player can leave the game at any time

by explicitly exiting or just by terminating the program. The player should also be considered as leaving if disconnect too long time.

2.2 Shared States

Our protocol maintains the following shared states:

1. All rats' name, positions and directions, their missiles' position. These states are trivial. If a rat leaves, the states associated with it is deleted.

2. A hit count matrix H. 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. If a rat i leaves, the row i and column i of H should be zero out after setting the base scores accordingly.

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

3. All rats' base scores, a vector B. 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. Say A hits B once and B hits A twice, and now B leaves, then base score of A is 1*11-2*5=1, which means A got 1 point from its interaction with B.

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:

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

As H and B can uniquely determine all rats' scores, the scores no longer need to be shared. And we choose H and B as our shared states instead of scores after thorough consideration. Since H and B can uniquely determine scores while scores cannot uniquely determine H and B, if we store scores, it is very difficult to recover the correct scores once some events are dropped by the network. For example, if A thinks A hit B, but B thinks A did not hit B, and if we use scores as shared states, we may finally come to a state where A's score increases by 11 but B's score remains unchanged, or vice versa. This is not good. Using hit count as shared states can avoid these kinds of cases.

2.3 Local Computation and Visualization

Each rat i locally computes its own position and direction, its missile's position, the off-diagonal elements of the hit count matrix H plus H_{ii}, the whole base score vector B and all rat's score. Those shared states that are not locally computed must be update from the received packet. Those locally computed states must also change to the corresponding values in the received packets.

Always show what should be show in the maze according to the states the local copy of the states. Since the local copy of the states may not be the most up to date, the visualization may not be correct temporarily before the local host receives update packet. But in this way, what the local host shows is

always consistent with what it knows. For hit event, if rat A sees it hits B, then it updates hit count, calculates scores, erases the missile and erases B. At this point, B is at nowhere in the maze from A's view. After receiving new packet from B, A will know the new position of B. If A sees that it is hit by B, then update those states, erase the missile and then reappear at a random available place.

2.4 Communication and Conflict Solving

The goal of communication is keeping the shared states consistent. In our protocol, each player i will send out packet containing its own position $< x_i, y_i >$, direction d_i, its missile's position $< xm_i, ym_i >$, its base score B_i, number of times it hit each other and number of missiles it launched h_i (the ith row of H). This gives us a very simple conflict solving method. That is always obey what the received packet says.

Our conflict solving method is very simple, but it is carefully defined after thorough consideration about almost all cases that can happen in a stable or unstable network. We will describe it in details in section 9.

2.5 No ACK

In order to guarantee the real-time playing, we use no ACK in our protocol. If we use any kind of ACK policy, like something in the TCP, it means that we need to wait for acknowledgement and retransmit some packets before we can make decision on some states. It will affect the real-time property of the game because all the packet or acknowledgement may be dropped or delayed by the network and keeps the player waiting for some time. Thus, we use no ACK in our protocol.

3 Packet Definitions

3.1 Packet Header

Type: 8bit
UserID: 8bit
Reserved: none

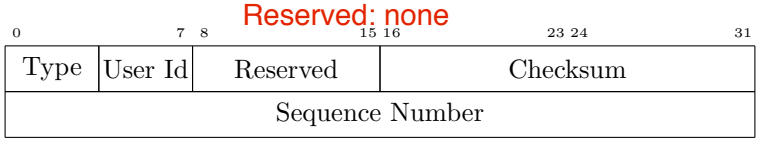

Figure 1: Packet Header

The first 4 bits indicates what kind of packet it is. There are four types of packets. The next four bits is user id. Since this game supports 8 players, 3 bits are enough (we used 4 bits in case we expand it to support 16 players). Following the user id is 4 reserved bits, then the checksum of the whole packet, and then is the sequence number, which monotonously increase as the player

Value	Type
0	HEART_BEAT
1	NAME_REQUEST
2	NAME_REPLY
3	GAME_EXIT

Figure 2: Packet Types

sends the packets. The sequence number shall wrap around after getting the maximum number.

3.2 Heart Beat Packet

0 7 8 15 16 23 24 31	
Packet Header	
Rat Position X	Rat Position Y
Rat Direction	Score Base
Projectile Position X	Projectile Position Y
Hit_Count[0]	Hit_Count[1]
Hit_Count[2]	Missile_Count[3]
Hit_Count[4]	Hit_Count[5]
Hit_Count[6]	Hit_Count[7]

Figure 3: Example of Heart Beat Packet Assuming This is Player[3]

The meaning of the field can be explained by its name. The last 16 bytes are based on our hit count matrix H. Where $H_{ij}, (i \neq j)$ counts the number of times i hit j. H_{ii} means total number of missiles i has launched. The hit count is -1 if that player doesn't exist.

Note that in the heart beat each player is sending one row of the H matrix. The row number corresponds to its player id.

3.3 Name Request Packet

The target id is the id of the target player. the player name field contains the name of the local player, ending in 0. In this way the player will also send out its own name when requesting others' names because others probably also do not know its name.

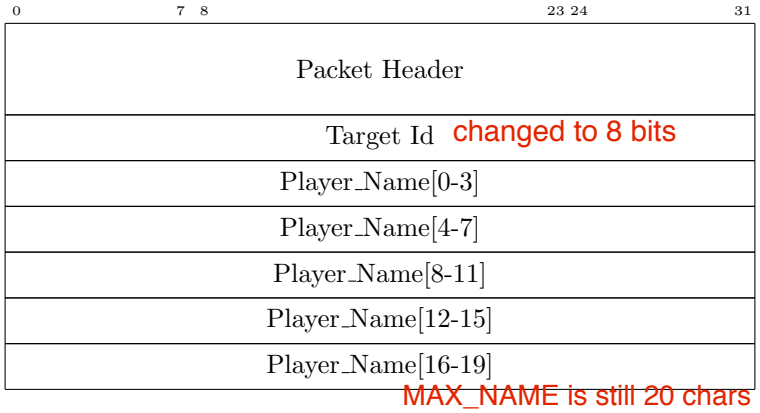

Figure 4: Player Name Packet

3.4 Name Reply Packet

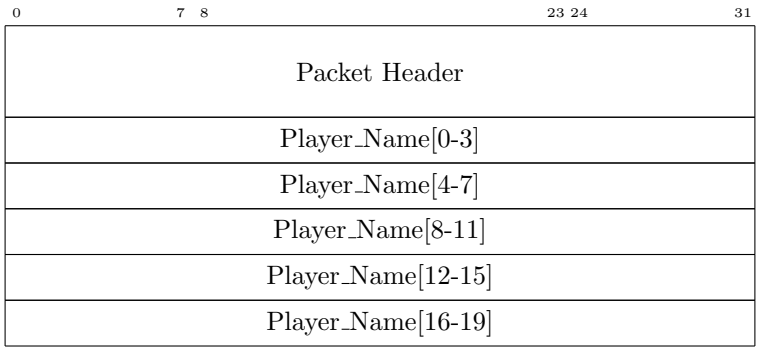

Figure 5: Player Name Packet

Roughly same as request name packet. The player name field contains the name of the local player.

GameExit

3.5 EXIT_GAME Packet

The format of leave packet is the same as the heart beat packet, because we want to utilized the EXIT_GAME packet to do another sync. The way to tell if it's a leave packet is by looking at the type value.

4 Game Playing Walk-through

4.1 Game Enter

4.1.1 User Id Selection

3 seconds

When a new player enters the game, he will first listen to the packets for 5 seconds, making sure that he has collected other players' heart beat. The new player gathers the Hit_Count array from all players and use it as a bit mask to determine which user slot is empty(1 means slot occupied and 0 means slot empty). He should bit-wise OR all the masks and pick out the first 0 position (empty slot) and use that slot as its user id. (for the first user, after 5 seconds he'll simply pick the first slot). After that he begins to send out heart beat packet with 0 value in the Hit_Count array.

4.1.2 User Name Request

An old player, which means it is not in the user id selection mode, receives heart beat packet with new user id, it periodically sends out a name request packet with the target id equals this new user id, together with its own name, until it receives the corresponding name reply packet. A player detect an user id as a new user id if in its hit count matrix the elements in the associated column are all -1. detect by seeing if the M->maze_rat[i].isPlaying == false

4.1.3 User Name Reply

When a player receives a name request packet targeting itself, it retrieves the sender's name in the packet. Then it initializes itself as a new player no matter whether it is true or not(consider the special case when it lost connection with the rest but got reconnected later). Then it sends out the name reply packet putting its name in it. Once a player receives a reply name packet, it updates the name and add this new player into the game by initializing that player's states.

From the above, it is designed that the new player will not send out name request packet. It knows all others' names by retrieving the names from the name request packet targeting it. And the new player will join the game when all players stop sending name request packet targeting it.

4.2 Game Exit

There are two mechanisms to detect game exit, one is by GAME_EXIT packet, the other is by detecting EXIT_TIMEOUT 3000

4.2.1 detect by GAME_EXIT packet

The player exits by sending out a GAME_EXIT packet. After sending that, it exits immediately.

Useful info in the GAME_EXIT packet is the exiting player id. It also comes with a heart beat info because it may give everybody a last sync up. On receiving a GAME_EXIT, the player will do three things:

1. First it should do the same routine as HEART_BEAT packet to sync up the states, as described in the next section.
2. Next it should recalculate its *base* vector based on the updated H matrix.
3. Finally it should invalidate the exiter's row and column in the H matrix (by setting the entries to -1)

4.2.2 detect by EXIT_TIMEOUT

If some player haven't received HEART_BEAT from a certain player for a certain amount of time, 10s in current version, it regards that player has left. It do the step 2 and 3 as in section 4.2.1.

4.3 Heart Beat Mechanism

To maintain consistency, each player should periodically send out HEART_BEAT packet every 200ms. The content includes its own location and direction, its missile's location, and its row of the H matrix.

When a player receives a HEART_BEAT packet, it shall update its local copy of shared data according to the new packet regardless of what is in the local copy.

4.4 Missile Tagging Detection

For any player, it only detects whether its missile hit others and whether it gets hit by others' missile, by comparing the position of the missile and the position of the rat. If hit event happens, it update the visualization accordingly, update its H-matrix and update scores. If the event is local player being hit, then it reborn at a random available place. Then send out heart beat packet.

5 Corner Cases

5.1 Packet Loss

If we lose a packet, heart beat mechanism will guarantee synchronized state again after a while, so in the end the scores and locations will be consistent.

5.2 Out-of-Order Packet Delivery

We track for the last packet's sequence number of each rat, if the new packet's sequence number is smaller, we discard the new packet. The sequence number is allowed to wrap around. In this way it can support maximum $2^{32} * PacketLength$ network delay.

5.3 Location Collision Handling

Two rats can appear in one cell due to temporary inconsistency. When rat A find it is in the same cell with B, A just pick a random neighbor available cell and jump in. It can happen on both sides of A and B. Then they will both choose new cell respectively and then update to each other by heart beat packet. If they collides again, then repeat previous steps until collision is solved.

5.4 Hit Conflict Solving

Due to network problem, rat A may see it hit rat B on its machine, but B do not agree from its view. For example at this time $H_{AB} = 3$ on A's machine but $H_{AB} = 2$ on B's machine. Then at some time later when network works well, B will receive the value of H_{AB} from A and B must obey that information. B will recompute score of A and B after updating $H_{AB} = 3$. And thus the scores will be consistent.

Since the times rat A hits each other is decided by A's view, the position of A's missile is decided by A, A's local calculation is always consistent with what A knows and local calculation is always correct regardless of the network, each missile A launches will never hit two rat. Besides, A cannot continue hitting B when B disconnect after hit him once and B's position is unknown at A's view.

As is stated in the section 2.3, visualization is decided by the local copy of states. If A hit B at A's view but A did not hit B at B's view, then A erase B, but B will not erase itself. When B sends heart beat, A will find that B didn't reborn at a new random place but a place that is close to where B dies. This is acceptable, since it does not affect playing and scores.

5.5 Id Conflict Solving

When receiving a packet with the same user id as myself, the player detects an ID conflict. The player first compares the sequence number. If the player's sequence number is larger than the received packet sequence number, it continues. On the other case if its sequence number is smaller, it initialize all its states and go to game enter procedure as a new player. This is because if its sequence number is smaller, all its packet will be regarded as some delayed packets and be dropped by others.