Mazewar Protocol Specification

Sining Ma and Tony Pan

April 20, 2014

1 Identifying Players

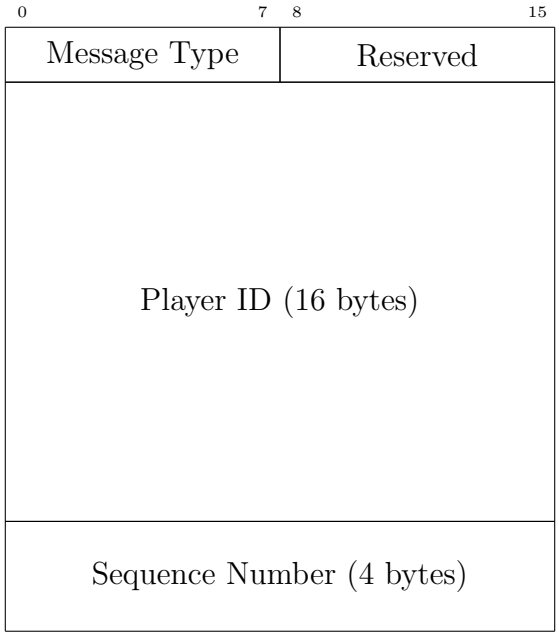
Players are responsible for identifying themselves with a Version 4 (random) UUID, which should be generated using the host machine's random number generator. Over the network, the player must transmit all UUID fields in the host's byte order (regardless of little- or big-endian). No conversion to network byte order is necessary because (1) the byte order of a UUID does not affect its randomness and (2) UUIDs do not need to be interpreted using the same byte order *across* hosts - individual hosts only need to interpret them in a *consistent* byte order.

2 Packet Types and Formats

This section details the binary layout of Mazewar packets. All fields are in network byte order with the exception of UUIDs (see Section 1).

2.1 Common Header

All Mazewar packets begin with a header of the same structure.



Message Type Each type of packet in Mazewar is assigned a unique message type identifier.

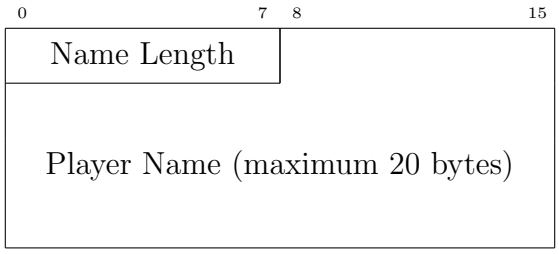
Reserved This field is reserved and should always be zeros.

Player ID Each player has a UUID as its unique identifier.

Sequence Number Each outgoing packet from a player should contain a monotonically increasing sequence number.

2.2 JOIN (Message Type 0xE0)

A player sends a JOIN packet to announce participation in the game.

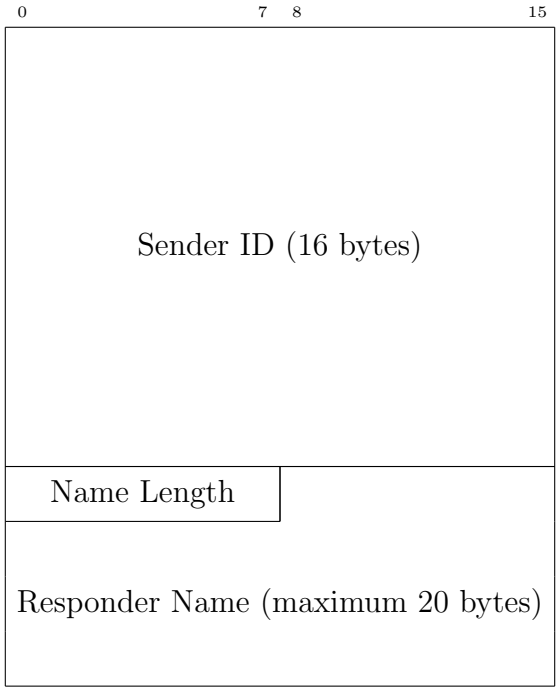


Name Length The length of the Player Name field. The maximum value is 20.

Player Name The player's name. It should not contain any NULL characters.

2.3 JOIN-RESPONSE (Message Type 0xE1)

A player sends this packet in response to JOIN packets.



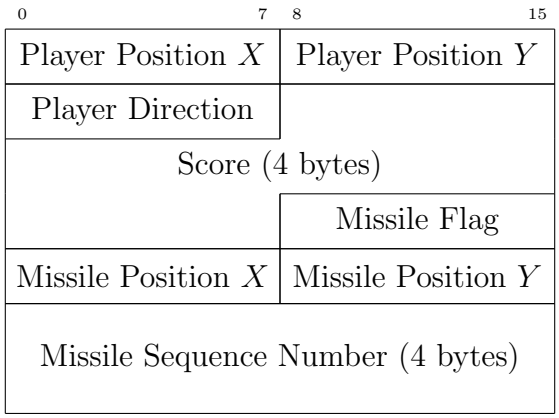
Sender ID The ID of the player who sent the corresponding JOIN message.

Name Length The name of the responder. The maximum value is 20.

Responder Name The responder's name. It should not contain any NULL characters.

2.4 KEEPALIVE (Message Type 0xE2)

Players periodically issue this packet to keep other players in the game updated.



Player Position X, Player Position Y The position of the player in the maze.

Player Direction The direction the player is facing. 0 is North, 1 is South, 2 is East, and 3 is West.

Score The score of the player.

Missile Flag When this field is zero, the player does not have a missile in flight, and the recipient should discard the next six bytes. When this field is one, the next six bytes specify the state of the player's missile. A player may only have one outstanding missile at any time.

Missile Position X**, Missile Position** Y The position of the missile in the maze.

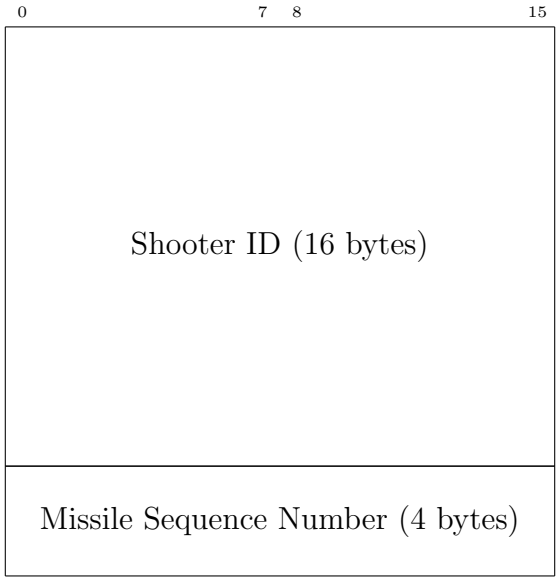
Missile Sequence Number The sequence number of the missile.

2.5 LEAVE (Message Type 0xE3)

A player sends this message to explicitly exit the game. This packet does not have any payload.

2.6 HIT (Message Type 0xE4)

The message is sent when a player determines that it has been hit by a missile.

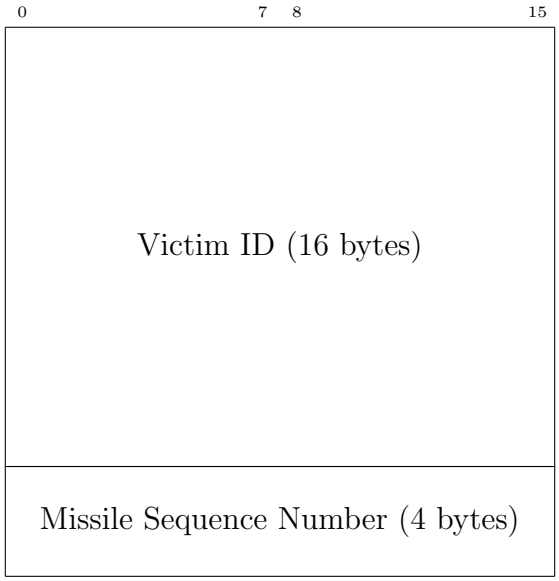


Shooter ID The ID of the player who fired the missile.

Missile Sequence Number The sequence number of the missile.

2.7 HIT-RESPONSE (Message Type 0xE5)

The message is sent by a shooter in response to a HIT message.



Victim ID The ID of the player who was hit.

Missile Sequence Number The sequence number of the missile.

3 Detailed Semantics

3.1 Joining the Game

3.1.1 The Join Phase

On startup, the program obtains the Player Name from the user. It should also generate a UUID to use as its Player ID (we do not need provisions in the protocol to handle conflicting IDs because we do not expect conflicting UUIDs).

Using these information, the program will generate a JOIN packet. The program now enters the Join Phase, which will last for three seconds. During the Join Phase, the program announces its presence and builds up its shared state. Specifically:

- Every 300ms, the program issues a JOIN packet to the multicast group.
- On receiving a JOIN-RESPONSE packet intended for itself (by inspecting the Sender ID field), the program stores the ID-Name pair extracted from the packet. If the associated name for an ID already exists, the new name will overwrite the old one.
- On receiving a KEEPALIVE packet from another player, the program stores the state of the player and its missile (if any), but does not display it on the GUI yet. The program only accepts KEEPALIVE packets with a larger sequence number than the most recent packet's sequence number with the same Player ID.

The program ignores all JOIN-RESPONSE packets outside the Join Phase.

At the end of the three seconds, the program will inspect the ID-Name pairs and ID-State pairs it collected during the JOIN phase. Any IDs without an associated name will be assigned the name "Unnamed".

Finally, the program places its rat on a random, unoccupied location on the maze, with a random, non-wall-facing direction. The program has now built up its view of the shared state and is now playable by the user.

3.1.2 Responding to JOIN

On receiving a JOIN packet, the player should extract and store the ID-Name pair of the new player. If the associated name for an ID already exists, the new name will overwrite the old one. The player should then send a JOIN-RESPONSE packet with the Sender ID filled with the new player's ID (even if it has already responded to a JOIN packet from that player earlier).

3.2 Leaving the Game

When the program exits the game, it sends a LEAVE packet to the multicast group and stops sending any other packets. On receiving a LEAVE packet, the program should remove the quitting player from its state. We do not need to handle lost LEAVE packets because a player is considered to have quit the game if others have not received KEEPALIVE packets from it for more than 10 seconds (Section 3.3.1).

3.3 Movement

3.3.1 KEEPALIVE

Players should issue KEEPALIVE packets every 200ms, the same frequency as missile movement.

When the user manually changes the position/direction of the rat, or fires a missile, the program immediately issues a KEEPALIVE packet.

If Player A has not received any KEEPALIVE packets from Player B for more than 10 seconds, Player B is considered to have left the game, and Player A should remove Player B and its associated data from the GUI and shared state.

Note that in practice we will not need to handle sequence number wraparounds - assuming 10 messages (all types) per second, the sequence number will last over 4000 hours.

3.3.2 Location Update and Conflict Resolution

Before moving, a player must ensure that its next move is valid based on the its view of the current shared state. Similarly, before respawning the player must make sure that the new location is valid.

Because of the distributed nature of the system, players must also handle situations where two players occupy on the same cell. For example, this can occur if two or more players simultaneously decide to move to an unoccupied cell. In Mazewar, the responsibility of location conflict resolution falls on the player with the smaller ID.

When Player A receives a KEEPALIVE packet from Player B, it first inspects the Player B's location. If B landed on the same location as A, and A has the smaller ID, A must move its rat to the nearest (in terms of Euclidean distance) unoccupied cell and immediately send a KEEPALIVE packet. If there are multiple such cells, it may choose any cell.

4 Missile and Scorekeeping

In Mazewar, the responsibility of missile tracking is divided between the "shooter" and the "victim". The player who fires the missile is responsible for updating its trajectory and lifetime (via KEEPALIVE). Players are also responsible for tracking whether they have been hit by another player's missile.

4.1 Missile Launch

Each player is required to maintain a Missile Sequence Number that monotonically increases for each missile fired, and is allowed at most one outstanding missile at any time.

4.2 Victims

When a player finds that it has been hit by a missile, it enters the Hit Confirmation phase. The phase ends when the player receives a HIT-RESPONSE or if two seconds have passed. During this phase, the player should suspend its missile hit test (the Hit Confirmation phase must not be reentered), but it should issue other packets (e.g. KEEPALIVE) as usual. This ensures that the player never claims to be simultaneously hit by multiple missiles.

If no HIT-RESPONSE is received during the Hit Confirmation phase, the player proceeds normally as if it has not been hit.

When the victim receives a HIT-RESPONSE intended for someone else for the same missle that it is waiting a confirmation for (judged by Missile Sequence Number), the victim exits the Hit Confirmation phase immediately.

When the victim receives a HIT-RESPONSE intended for it, it updates its score and respawns in a new valid location and direction. It must also immediately sends a KEEPALIVE message.

4.3 Shooter

All players are required to maintain a Missile Sequence Number to (Player ID, Timestamp) mapping. On receiving a HIT message intended for its missile, three possibilities exist for the mapping:

Missile Sequence Number does not exist
: The player must issue a HIT-RESPONSE to the victim and update the mapping with the current timestamp. It should also update its own score. The player must then increment its Missile Sequence Number counter. Finally, the player should clear its outstanding missile and immediately send a KEEPALIVE packet with the Missile Flag cleared.

Missile Sequence Number exists but maps to a different Player ID

This means that multiple players are claiming to have been hit by the same missile. The player must ignore this packet.

Missile Sequence Number-Player ID mapping exists

The player must issue a HIT-RESPONSE again. It does not need to increment the Missile Sequence Number counter.

To control memory footprint, all map entries must timeout and be removed after five seconds.