<Assignment #1: Mazewar>

Protocol Design and Specification

Team KSR
Hong Zhang (romanzhg@stanford.edu)
Kyuhwan Dhong (kyuhwan@stanford.edu)
Lei Sun (sunlei@stanford.edu)

1. Introduction

Mazewar is a distributed, multiplayer game that allows each player to control a rat in a maze. A player receives points for tagging other rats with a projectile and loses points for being tagged.

- The protocol is completely distributed without central point of failure
- The communication is over UDP multicast
- The protocol deal with lost packets, packet reorder and packet delay
- We assume that players don't send out malicious or fraudulent packets

2. Shared State

The client keeps listening all traffic in this game, and then updates its game state accordingly. These are some of the agreements between players:

- All data shared across the network are in network (i.e. big endian) byte order unless otherwise specified.
- The maze size is 32x16. This is the same as the given default.
- Projectile moves 1 cell per 100 milliseconds (10 cell/s).
- The cooldown time for a projectile is 2 seconds. In other words, a projectile is limited to 1 per 2 seconds for every player.
- Players do not know the position of other player's projectile.
- Maximum number of players is 10.

The shared states between players are:

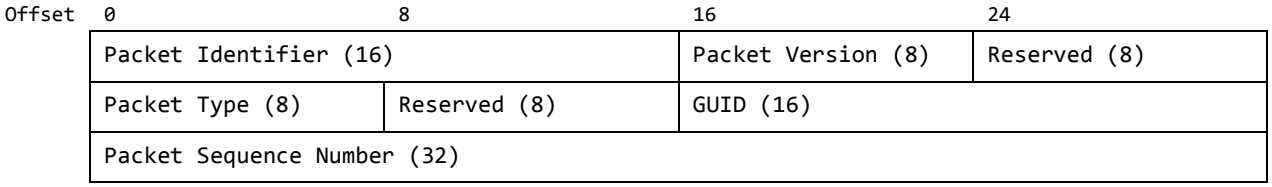
- **The position and direction of each player.** This is shared between players every 500ms, regardless of the rat's movement.
- **Current score.** Since we trust all players' behavior we trust the score they share every time.
- **User name of each player.** The length of the username is 24 characters maximum.
- **Projectile tags.** We do not share the projectile position itself.

3. Packet Types

Every packet is 36 bytes in size (12 bytes for the header and 24 bytes for the descriptor).

Value	Descriptor	Reader	Description
0x01	STATE	All	Contains position and direction information, as well as current score
0x10	USERNAME-REQUEST	Specified by GUID	This response contains the user name and GUID
0x11	USERNAME-REQUEST-ACK	Specified by GUID	Sender request the receiver's username
0x20	TAGGED	Specified by GUID	Informs some other player that "You are tagged by my projectile"
0x21	TAGGED-ACK	Specified by GUID	Acknowledges back to the shooter that whether he agrees or disagrees with the tag claim.
0x30	LEAVE	All	Tell other players that I am quitting this game.

4. Packet Header



Packet Identifier (16)

A packet identifier to identify whether this UDP packet belongs to mazewar. The value is set to 0x4d57 which is MW in ASCII. All other packets that differ to this identifier should be dropped.

Packet Version (8)

Version number in 8-bit integer. The version number starts from 1 and increments whenever necessary. For now, 1 is the only packet version that exists.

Packet Type (8)

Indicates the type of the packet, which is one of the types defined in section 3.

GUID - Global Unique ID (16)

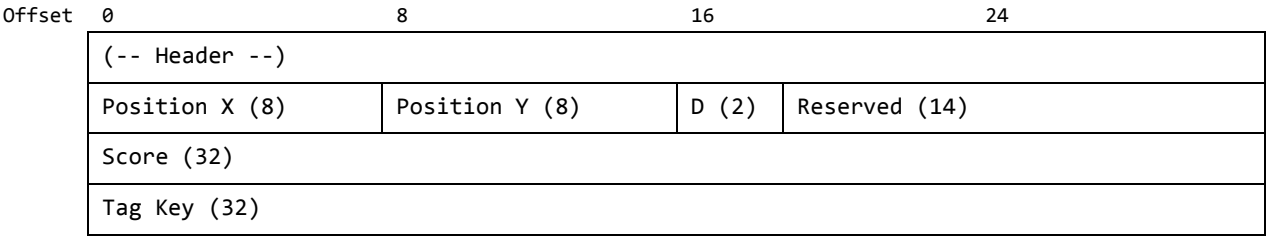
A randomly generated 16 bits number assigned to each player. It is unique to each player, collision will be resolved, as explained in later section.

Packet Sequence Number (32)

32 bits number to identify each packet. It is incremented every time a packet is sent out. Note that this sequence number is not global and is maintained individually by each player. Local client should keep the largest sequence number from each other player and determine any packet re-ordering. Packet sequence number starts from 1 and is incremented after each packet send.

5. Packets

STATE (0x01)



Note that we drop any STATE packets that are lower than the latest STATE packet sequence number.

Position X (8)

Absolute value of the X position. 0 is the leftmost column's value.

Position Y (8)

Absolute value of the Y position. 0 is the uppermost row's value.

D – Direction (2)

2 bits is used to indicate the direction this rat is facing, the value of which is shown in the table below.

Directions	Value
NORTH	0
SOUTH	1
EAST	2
WEST	3

Score (32)

A signed value to represent the current score of this player

Tag Key (32)

A sequence number start from 1. It is incremented after each tag event.

USERNAME-REQUEST (0x10)

Offset	0	8	16	24
	(-- Header --)			
	GUID (16)		Reject (1)	Reserved (15)

This packet is sent out whenever a STATE packet with an unknown GUID is received.

GUID (16)
GUID of the unknown player.

Reject (1)
Reject value 1 stands for a proposal of rejecting the player. Value 0 stands for accepting the player.

USERNAME-REQUEST-ACK (0x11)

Offset	0	8	16	24
	(-- Header --)			
	Username (192)			

This packet is sent out when a USERNAME-REQUEST packet is received with his GUID. To avoid burst duplicate USERNAME-REQUEST-ACKs, the player sends this packet only if he didn't send this in the last 500 millisecond.

Username (192)
24-byte username in ASCII. The limit for username is 24 ASCII characters. Note that the string value may not be a null-terminated string.

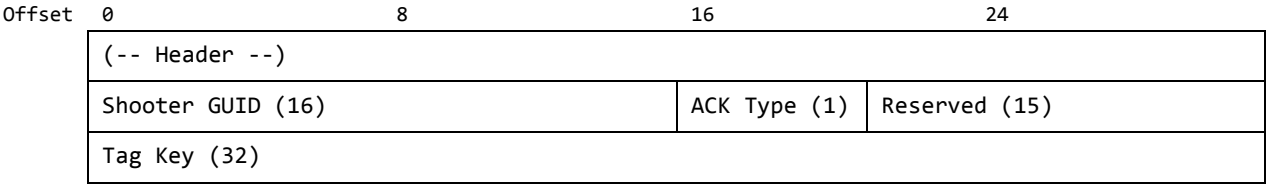
TAGGED (0x20)

Offset	0	8	16	24
	(-- Header --)			
	Tagged Player GUID (16)		Reserved (16)	
	Tag Key (32)			

Tagged Player GUID (16)
Tagged player's GUID.

Tag Key (32)
Tag key value of the player. This should be the latest tag key that the owner of the projectile knows.

TAGGED-ACK (0x21)

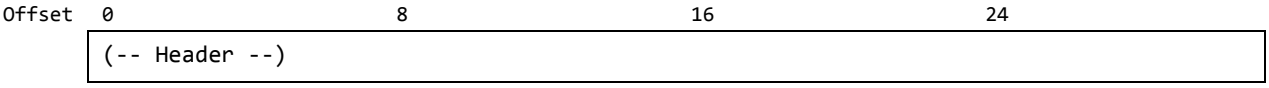


Shooter GUID (16)
Shooter's GUID.

ACK Type (1)
ACK type with value 1 stands for ACK, value 0 stands for negative ACK.

Tag Key (32)
The Tag Key from the TAGGED packet that we are acking on.

LEAVE (0x30)



6. Protocol Semantics

State Update
Every player sends STATE and other event generated packet when

- Local location or direction changed.
- After 500 milliseconds even if there is no state change.
- After a re-spawn.

Join the game
Each player maintains a list of GUIDs of other players in the game. If a player received a packet with a new GUID, it will sent out USERNAME-REQUIEST packet to request the username, when it receives

the USERNAME-REQUEST-ACK from that GUID, it will add the other player to its list, the joining process finish.
A player received a USERNAME-REQUEST with its GUID stated, should send out a USERNAME-REQUEST-ACK in response. However, the player does not need to send another USERNAME-REQUEST-ACK in case it sent out one in the past 500ms window. This is to avoid unnecessary burst ACKs to other players.

The new player will send USERNAME-REQUEST for every existing player since it will not know about other users and would need to know the username.

Example:
Suppose A is the only player and B just started the game, so B is going to join. B first generates its 16-bit random GUID number. Then B will send out the STATE with this GUID.

From A's point of view, a packet with an unknown GUID appears, it sends out a USERNAME-REQUEST packet, when A receives the USERNAME-REQUEST-ACK from B, it accepts B to his game.
From B's point of view, since A is keep on multicasting it's STATE, and A's GUID is unknown to B, B will also send the USERNAME-REQUEST packet, upon A's response, B can add A to its game.
The joining process is finished.

Rejoin the game

Due to network problem a player may temporarily lost connection to other players, the joining of such player is called rejoin. The process of rejoin is the same as join at the first time. The dropped player will remember his status and retain its score.

GUID Collision

When a player received a packet from another player with the same GUID as its own, this is considered as a GUID collision. The player will re-pick a random GUID. The remaining sequence is identical to rejoining the game. Eventually, all players will have a unique GUID. Also, note that this is very rare to happen. Please read the rationales for more detail.

Because UDP multicast packets loopback to the sender, packet sequence number is also checked to distinguish between the loopback and received packets. Whenever a packet is received it checks the packet sequence number. If the GUID is same as its own but the packet sequence number is greater than the last value, this packet is considered as loopback. If the GUID is same as its own and the packet sequence number is less or equal than the last value it is considered as GUID collision. Note that this will make both players to select a new GUID when a GUID collision happens.

Example:
Suppose A and B is in the game and C just started, and C happen to have the same randomly generated GUID as B. For both B and C, they will receive a STATE packet with the same GUID as its own, so both of them re-generate a 16-bit GUID randomly.

From A’s point of view, B rejoin the game.

Game Capacity and Reject

Each player maintains a list of other players decision (the reject bit in USERNAME-REQUEST packet). The player will decide if itself needs to leave after 2 seconds every time it receives a USERNAME-REQUEST packet. If the number of reject is equal or more than the number of accept, the player will conclude that the game is already full. Note that a player does not count itself in this decision.

When a player decides itself to leave due to the game full, it will go into a re-connect mode. A player will generate a random number between 500ms and 1000ms and reconnect after this amount of time. It this attempt fails 3 times in a row, the player will leave with all its current state lost.

State Collision

If a player receives the same position as itself for 3 or more consecutive STATE packets, it is considered as a collision. This player will randomly move to a random valid position in 1 cell distance.

Tag Key

Tag Key starts from 1 and increments every time it gets re-spawned. We use the tag key to avoid simultaneous projectile tags. Details are described in the Tag Behavior section.

Projectile Behavior

In our game, each player can fire a projectile every 2 seconds, the projectiles disappear immediately when it hits wall.

Projectile information is kept locally, which means only the player who fires the projectile will know its position and whether it tags another player.

Tag Behavior

When a user believes he tagged another player, he will send out a TAGGED packet with the tag key of the tagged player. The player will send TAGGED packet every 500ms until it receives TAGGED-ACK or the player is considered left. It increases its score only after receiving a positive TAGGED-ACK from the tagged player.

The tagged player subtracts its score when the TAGGED packet matches its tag key. Then this player will send a positive TAGGED-ACK. All other TAGGED packets are responded with a negative TAGGED-ACK. Concerning a packet loss, the tagged player will retain any positive tagged information in a history table(in this case we don’t subtract the score). It positively ACKs TAGGED packets inside this history table as well. A player will save this tagged information up to 5 TAGGED information for each player.

Example:
If X and P claim they tagged Y at the same time and X's packet arrives first:

- Y processes X's TAGGED packet. The tag key matches its current tag key so it considers itself tagged. Subtracts 5 points.
- Y sends a positive TAGGED-ACK to X, and go to the re-spawn process. During the re-spawn process it will regenerate the Tag Key. Also, Y saves X's tag information.
- X receives a positive TAGGED-ACK. He adds 11 points to its score and stops re-sending the TAGGED packet.
- T processes P's TAGGED packet. This time the tag key does not match its current tag key so Y considers this as a no hit.
- Y sends a negative TAGGED-ACK to P.
- P receives a negative TAGGED-ACK. He ignores the tag and stops re-sending the TAGGED packet.

If Y's TAGGED-ACK packet gets lost, X will still try to resend its TAGGED packet. When Y receives this TAGGED packet it will find it in the TAGGED history and send a positive TAGGED-ACK, but not subtracts its score this time.

Re-spawn

Re-spawn happens when a player

- Joins the game
- Tagged by a projectile

During re-spawn, player will be located at a random valid position, not facing a wall.

Leave Game

A player sends LEAVE packet right before he leaves the game. All other player that receives this player's LEAVE packet will remove this player's information immediately. If the LEAVE packet gets lost this player will fail over to disconnection and eventually all players will know this player has left.

Disconnection

If players do not receive a STATE packet from another player for more than 10 seconds, we consider this player has left the game or disconnected. All states for the player will get removed locally.

7. Rationales

Sequence number

Suppose a client sends 10 packets per second, 32 bits can be used for (2^32-1) / 10 / 3600 / 24 / 365 = 13 years, which should be enough.

Player GUID

Player GUID is a 16-bit(65536) unsigned integer and the maximum number of players is set to 10. The possibility that a collision can happen is 10/65536(= 0.00015258789) which is rare enough Also we do have a collision resolution mechanism. So, in any case that it happens, things will get resolved eventually.

Game Capacity and Reject

The number of player in a game should be limited. To ensure game quality, the game definitely should not support as much players as the capacity of LAN. Moreover, the game capacity will affect the rate of GUID collision.

The max number of players is set to 10.

Number of Tagged Packets to Remember

A player has to remember certain number of TAGGED packets to deal with TAGGED packet retransmissions. Since there is no TAGGED-ACK-ACK, we have to decide when we will garbage collect this information. Otherwise, the number of saved TAGGED packets will grow indefinitely.

We have decided to retain 5 TAGGED packets per player. We decided with this number because of the following reasons.

- Each rat can fire every 2 seconds.
- A rat is considered disconnected if we don't receive a STATE packet for 10 seconds.
- This means there can be up to 5 shots accumulated without being ACKed.