CS244B Project 1

Wei Shi
weishi@stanford.edu

Avinash Parchuri
aparchur@stanford.edu

April 23, 2013

Contents

1 Protocol overview

1.1 Purpose

Mazewar is a distributed multi-player game. The Mazewar protocol defines a way for several Mazewar clients to communicate with each other.

1.2 Terminology

This specification uses a number of terms to define the Mazewar protocol.

Packet
: This is the basic unit of communication between two Mazewar clients. It is of fixed length, and consists of a packet header (defined in Section 2.1) and a packet body whose format varies depending on the type of the packet.

Client
: An application that implements the Mazewar protocol.

Kill
: When a missile launched by another client touches a client, that client is said to be killed by the client who launched the missile.

Victim
: A client that has been 'killed' by another client's missile. It is the responsibility of each client to determine whether it is the victim of a kill and if so, to also determine the killer.

Killer
: A client that has fired a missile that killed another client.

Game Event
: This refers to any event that causes a change in the state of the current game. This includes valid input from the player and network input from other clients.

Local Game Event

This refers to input from the player that causes a change to the local game state (ex: movement, change of direction, firing a missile).

1.3 Generic Grammar

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119.

1.4 Overall Operation

The Mazewar protocol is based on the multicast paradigm. Mazewar clients establish a communication with each other by joining a pre-defined UDP multicast group on a specific port. Any outgoing data is transmitted as UDP packets multicast to all members of that multicast group on that same port. The clients use this multicast to share their state with other clients in the group, thereby ensuring overall consistency. A request-response paradigm is also used for certain operations by embedding the responder's id in the packet data.

The packets are transmitted and received obeying the network byte order and clients are REQUIRED to convert incoming packets to host byte order and outgoing packets to network byte order.

Clients identify each other with a randomly generated 32-bit number which serves as a client-id. Sections 3.1 and 3.2 detail the steps taken to ensure uniqueness of these values.

2 Protocol specification

2.1 Packet Header

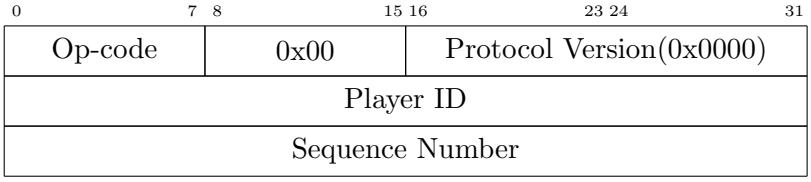

Figure 1: Packet header

Op-code	A byte uniquely identifying the packet type
Protocol Version	Version number of the protocol in this packet. It is reserved for future modification of the protocol. It remains 0 in this version.
Player ID	ID of the client that sends the packet. Randomly generated and unique among all players.
Sequence number	A counter for the packet sent. Wrap around to 0 upon overflow. Used to detect packet reordering.

2.2 Heartbeat

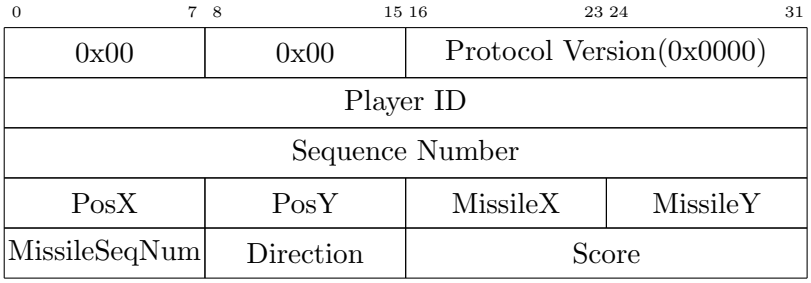

Figure 2: Heartbeat Data Packet

PosX	The X co-ordinate of this client's position in the maze
PosY	The Y co-ordinate of this client's position in the maze
Direction	The Direction that this client is facing
MissileX	The X co-ordinate of this client's missile. When this value is negative, the client does not have an active missile
MissileY	The Y co-ordinate of this client's missile. When this value is negative, the client does not have an active missile
MissileSeqNum	A sequence number of the active missile.

The Heartbeat packet is sent out by each Mazewar client at every game

event or timeout event. This is used by other clients to keep track of the position of the current player, as well as the position of the missile fired by the current player (if any).

The missile sequence number is used in Kill packet to identify the missile that causes the kill. The shooter only confirms one kill per sequence number to avoid the situation where multiple clients think they are hit by the same missile.

2.3 NameRequest

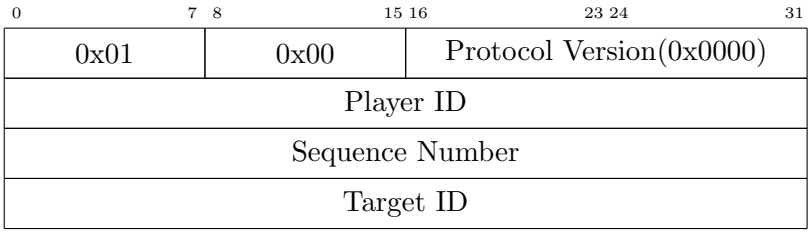

Figure 3: NameRequest Data Packet

Target ID ID of the client whose name is being requested

The NameRequest packet is sent out by clients when they start receiving Heartbeat packets with a new client ID.

2.4 NameResponse

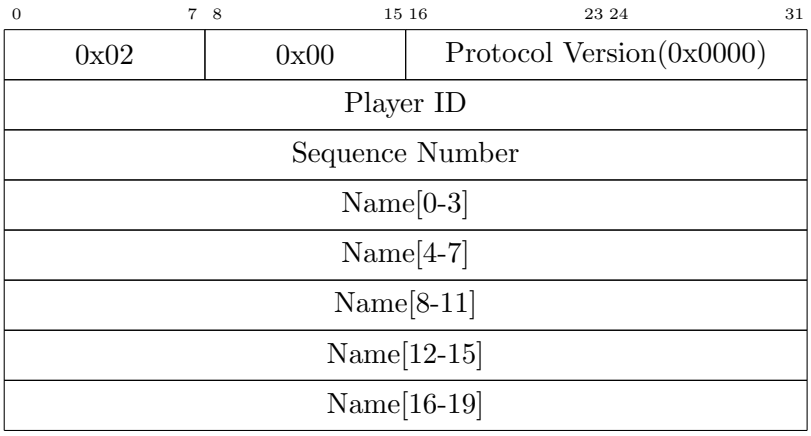

Figure 4: NameResponse Data Packet

Name Name of the sending client

The NameResponse packet is sent out by the client when it receives a NameRequest packet targeted at it.

2.5 Killed

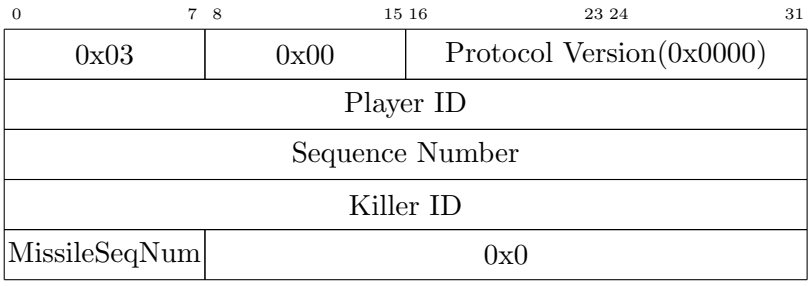

Figure 5: Killed Data Packet

Killer ID ID of the client that killed the sender
MissileSeqNum The sequence number of the missile that kills the client.

The Killed packed is sent out by a client when it detects that a missile fired by another client has hit it. When the client that fired the missile receives this packet, it will send a KillConfirmed packet and update its score and broadcast it in the next Heartbeat packet.

2.6 KillConfirmed

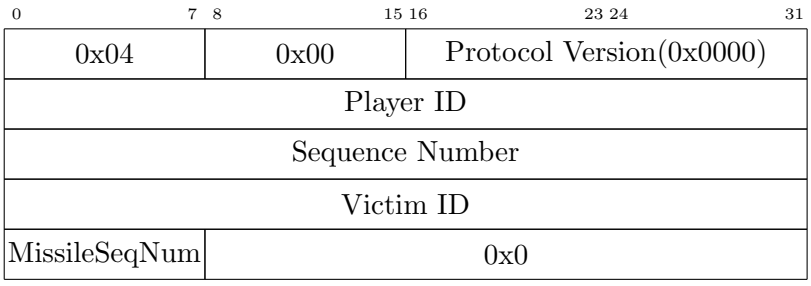

Figure 6: KillConfirmed Data Packet

Victim ID	ID of the client that is killed
MissileSeqNum	The sequence number of the missile that kills the client.

The KillConfirmed packed is sent out by a client when it receives the Killed packet from another client. If multiple clients think they are killed, it only confirms the first Killed packet. So other clients will register the kill.

2.7 Leave

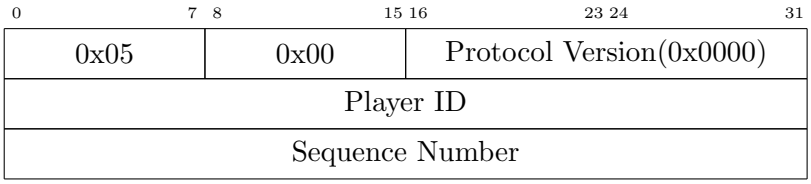

Figure 7: Leave Data Packet

The Leave packet is sent out by a client when it leaves the game. The client who receives this packet should remove corresponding player from the list.

3 Supported Operations

3.1 Joining a New Game

When a client starts up, it SHALL join the UDP multicast group. It SHALL then enter the join phase. While in the join phase, the client SHALL monitor the network for incoming heartbeat packets. The client SHALL remain in the join phase for 2 seconds. At the end of this period, the client MUST generate a random 32-bit unsigned integer that does not match any of the client-id's seen during the join phase. The client SHALL then move into the play phase.

3.2 Resolving ID Conflicts

By using a good seed for the random number generator, client implementations can bring down the chance of ID conflicts significantly. Suggested seeds include a combination of the network MAC address, time and process id of the client. In such a case, the probability of two clients generating the same id becomes negligible (in a 32-bit address space, the chance of collision is 5.4210109e-20). However, this protocol defines a simple way for clients to resolve id conflicts, so as to ensure absolute robustness. When a client in the play phase receives a heartbeat packet where the sender's client-id matches it's own, the client SHALL compare the sequence number in the received packet with it's current sequence number. If it is found that it's current sequence number is less than or equal to the sender's sequence number, the client SHALL leave the game (as described in Section 3.9), clear it's local state and join again (as described in Section 3.1).

3.3 Discovery of New Clients

Clients connected to the same port SHALL discover each other dynamically using by using heartbeat packets. A client that is in the play phase MUST send out a heartbeat packet (as described in Section 2.2) at the occurrence of a local game event, as well as at a regular interval of 200ms. When a client receives a heartbeat packet from a new client (as determined by the client-id in the packet), the client MUST update it's local state as necessary to ensure that game rules can be followed taking the new client's state into account. Clients MUST be able to support games with up to 8 other clients. Beyond this, it is left to the implementation to decide on how to handle extra clients.

3.4 Name Retrieval

This protocol allows clients to exchange null-terminated ASCII strings with each other to get display/user names for use in score-cards or elsewhere. The protocol limits the length of these strings to 20 characters (including the null-terminator) so clients with longer strings MUST truncate the strings before transmitting.

When a client discovers a new client (as described in Section 3.3) it SHALL send a NAMEREQUEST packet (described in Section 2.3) with the 'Target ID' field set to new client's id.

Upon receiving a NAMEREQUEST packet where the 'Target ID' field matches the client's id, the client MUST respond with a NAMERESPONSE packet (described in Section 2.4) with the 'Name' field set to the client's null-terminated user-name string. In the case of an implementation that chooses to not track names or to not send out a name, the client MUST respond with a empty, null-terminated ASCII string.

When a client receives a NAMERESPONSE packet, it MAY retrieve the name from the packet. It is left to the implementations to decide whether to use the string returned with a NAMERESPONSE.

In the case that a client sends out a NAMEREQUEST but does not receive a NAMERESPONSE, the client MAY choose to periodically reissue the NAMEREQUEST packet as long as the frequency of reissual is less than or equal to one NAMEREQUEST packet per second.

3.5 Movement

When a client wishes to make a move, it SHALL first check it's local state to determine whether the move is valid (i.e. that the resulting location of the move is unoccupied by any other client and is not a wall). If it determines that the move is indeed possible, the client shall update it's local state to reflect it's new position and send out a heartbeat packet to notify other clients. If, due to network or timing issues, a move results in two clients occupying the same location in the maze, the clients MUST resolve the conflict as described in Section 3.6.

3.6 Resolving Positional Conflicts

When a client receives a heartbeat packet from another client indicating that the other client is located in the same location, the client SHALL compare the client-id of the other client to it's own client-id. If the client's client-id is less than that of the other client, the client MUST try to move to

an unoccupied and available adjacent location. If such a location does not exist, the client SHOULD try to move to locations one unit further away, then two locations further away, so forth, till an unoccupied and available location is found. If multiple clients occupy the same location and some of them resolve into the same location again, this process will be recursively applied.

3.7 Launching Missiles

Each client MUST have at most one missile active at any given time. Once a previously launched missile hits another client or a wall, it is deemed inactive and the client MAY launch a new missile. When a missile is launched, the client MUST assign it a sequence number which will be transmitted in the heartbeat packets.

When a missile is launched, it's location and direction SHALL be the same as that of the launching client. An active missile SHALL move 1 position in the direction of travel every 200ms. The client that launched the missile is REQUIRED to update the missile's position as stated above in it's local state and then notify other clients of the change through the 'MissileX' and 'MissileY' fields of the heartbeat packet (described in Section 2.2). If a client does not have an active missile, it MUST set either of these fields to a negative value before transmitting the heartbeat. When there is an active missile, the client SHALL include the sequence number of the missile in the 'MissileSeqNum' field of the heartbeat packet.

A client MUST keep track of the status of at least the last two fired missiles. At a minimum, the client MUST track whether the missiles have killed anybody, and if so the client-id of the killed client.

3.8 Kill Detection

Each client SHALL be responsible for detecting whether it has been killed by another client's active missile. The determination MUST be done by comparison of the missile location from the heartbeat packets (Section 2.2) of other clients and the client's own location to see if they are equal. If it is determined that the client has been killed by a missile, the client SHALL send a 'KILLED' packet (described in Section 2.5) with the 'Killer ID' field set to the client-Id of the killer and the 'MissileSeqNum' set to the value from the Killer's heartbeat packet. In the case that a client determines that it could have been killed by multiple clients, the client MUST send the KILLED packet to only one of the potential killers. It is left to the

implementation to choose which killer to send the packet to.

When a client receives a KILLED packet from another client and finds that the 'Killer ID' field matches it's own client-Id, the client SHALL use the value in the 'MissileSeqNum' field to determine whether that missile had already killed another client. If is determined that the missile had not killed anyone yet or that the missile had killed the sender of the KILLED packet, then the client SHALL respond with a KILLCONFIRMED packet (Section 2.6) with the 'Victim ID' and the 'MissileSeqNum' fields set accordingly and then update it's score to reflect the kill. In this case, if the missile was previously active, the client SHALL make it inactive. If the missile had not previously killed anybody, the client SHALL mark it as having killed the sender. If it is determined that the missile had already killed another client, it SHALL respond with a KILLCONFIRMED packet with the ID of the killed client. If the missile in question is not being tracked by the client any longer, the client SHALL not send out a KILLCONFIRMED packet.

When the victim receives a KILLCONFIRMED packet it has been waiting for, it SHALL update it's score to reflect the kill and relocate to a randomly selected, unoccupied position. To account for sporadic network issues, the victim MUST keep sending KILLED packets to the killer with every heartbeat packet for at most 5 seconds. The victim SHALL stop sending these packets when it sees a KILLCONFIRMED from the killer for the same missile(targeted towards it or to another player) or when the killer leaves or times out (Please see Section 3.10). In the rare case that the victim has to wait a long time for a KILLCONFIRMED packet, the victim can operate as normal. However, it SHALL not respond to any KILLED packets or send out KILLED packets itself during this period.

3.9 Leaving a Game

When a client wishes to leave a game in progress, it SHALL send a LEAVE packet (Described in Section 2.7) to the UDP group. Upon receiving a LEAVE packet, a client SHALL immediately remove the sender from it's local game state. Any KILLCONFIRMED requests to the sender will be terminated at this point.

3.10 Detecting Abnormal Client Terminations

All clients SHALL make sure that they are receiving heartbeat packets from all other known clients on a frequent basis. If a client determines that it has not received a heartbeat packet from another client in the last 5 seconds,

the client SHALL remove the other client from it's local state.

4 Network Reliability Considerations

4.1 Re-ordering of Packets

Since UDP does not guarantee receipt of packets in the order in which they were transmitted, every client implementing the Mazewar protocol MUST include an unsigned 32-bit sequence number in the header of each packet. This sequence number is incremented for each packet sent out.

This number allows clients receiving the packet to determine whether it is outdated, and if so to discard it to avoid inconsistencies in the game state. Every client MUST check the sequence number of an incoming packet and if they find that the value is smaller than the largest sequence number received from that same client so far, discard the packet.

A 32-bit unsigned number gives a client 4,294,967,296 packets before overflow. Even assuming that clients generate 10 packets for second (this is very rare, since the normal broadcast interval is 200msec), clients will run for more than 13 years before overflowing. Therefore, no provision is made in the protocol to account for sequence number overflows.

4.2 Dropped Packets

Since clients are required to continuously broadcast their state to the Mazewar group, even when some packets are dropped, the group will maintain a consistent shared state. In some cases, dropped network packets might cause conflicts in positioning, but the protocol defines a way for these situations to be handled in Section 3.6 so that consistency is eventually reached.

In the case that two clients are required to agree with each other on a kill event, the clients make a best effort to reconcile, with the victim repeatedly sending out KILLED packets to the potential killer, to which the killer replies with a KILLCONFIRMED (provided the kill is valid). This prevents sporadic instances of dropped packets from interfering with the communication.