CS244B Mazewar Protocol Specification

Ned Bass <nedbass@stanford.edu>
Prakash Surya <surya1@stanford.edu>

April 13, 2012

1 Introduction

CS244B Mazewar is a distributed, multiplayer game that allows each player to control a rat in a maze. Each player receives points for tagging other players with a projectile, and loses points for being tagged and shooting projectiles. Due to it's distributed nature, the game is fault-tolerant, as players can continuously leave and join the game without disrupting other players.

2 Protocol Description

The Mazewar protocol defines the way in which each instance of the game communicates over the network, and is intended to be implemented over an unreliable transport layer such as UDP. In addition, the protocol assumes all players (including new players trying to join) see all Mazewar communication sent over the network via multicast. The protocol consists of two distinct phases of communication, and a set of well defined packet types.

2.1 The Discovery Phase

The discovery phase provides new players the ability to discover any currently active game. This phase is also used to learn the global game time, which is discussed in Section 4.8. While in this phase, the client does not send any outbound traffic and listens for a minimum of 5 seconds for any incoming Mazewar traffic from other active players. If there is no incoming traffic during this time, it is assumed there isn't a current game being played on the network.

2.2 The Active Phase

During the active phase a player is actively participating in a Mazewar game. While in this phase, each client may send any of the packet types defined in Section 3.

3 Packet Definitions

The following packet types are defined by the protocol:

Descriptor	Description
State	Communicates position and direction of the rat, position and direction of the projectile, and the player's score.
Nickname	Communicates the nickname and GUID.
Tagged	Sent when the local rat has been hit by a remote rat's projectile. This message must be acknowledged.
Tagged ACK	Acknowledge receipt of a tagged packet.
Leaving	Advertises a player leaving the game.
Request for Retransmission	Requests a client to retransmit a packet

3.1 Packet Header

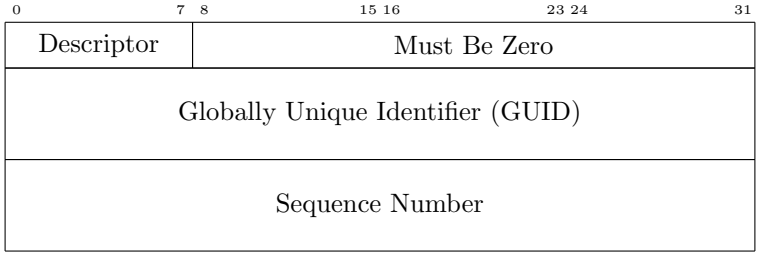

Figure 1: Packet Header

3.1.1 Descriptor

Value	Type
0	State packet
1	Nickname packet
2	Tagged packet
3	Tagged ACK
4	Leaving packet
5	Reqest for retransmission packet

3.1.2 Must Be Zero

A 24-bit field reserved for future use, such as for a protocol version number.

3.1.3 GUID

A randomly generated 64-bit identifier used to distinguish clients. This is determined upon joining a game and must not change during a single session. The probability of a GUID collision is assumed to be negligible so no provision is made for resolving conflicts.

3.1.4 Sequence Number

A monotonically increasing number that uniquely identifies a packet transmitted by a client. This must be incremented by one for each packet transmitted unless the packet is a retransmission. This can be used for detecting out of order packet delivery, dropped packets, and for requesting retransmission.

3.2 State Packet

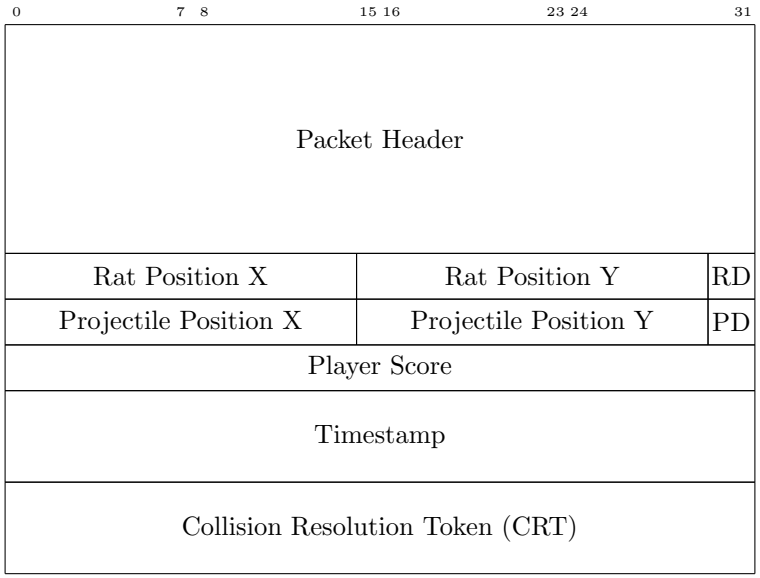

Figure 2: State Packet

3.2.1 Rat Position X, Rat Position Y, and Rat Direction (RD)

The maze coordinates of the rat and the direction it is facing.

3.2.2 Projectile Position X, Projectile Position Y, and Projectile Direction (PD)

The maze coordinates of the rat's projectile and the direction it is travelling. If no projectile is active, the word containing these fields must be `0xffffffff`.

3.2.3 Player Score

A signed value representing the player's current score.

3.2.4 Timestamp

Represents the global game time G at which the packet was transmitted. G is defined as

$$G = E + t$$

where E is the local epoch (see Section 4.8), and t is the number of milliseconds elapsed since E was initialized.

3.2.5 Collision Resolution Token

A 64-bit value randomly generated for each packet. This is used to resolve timing conflicts. See Sections 4.2 and 4.3.1 for further details.

3.3 Nickname Packet

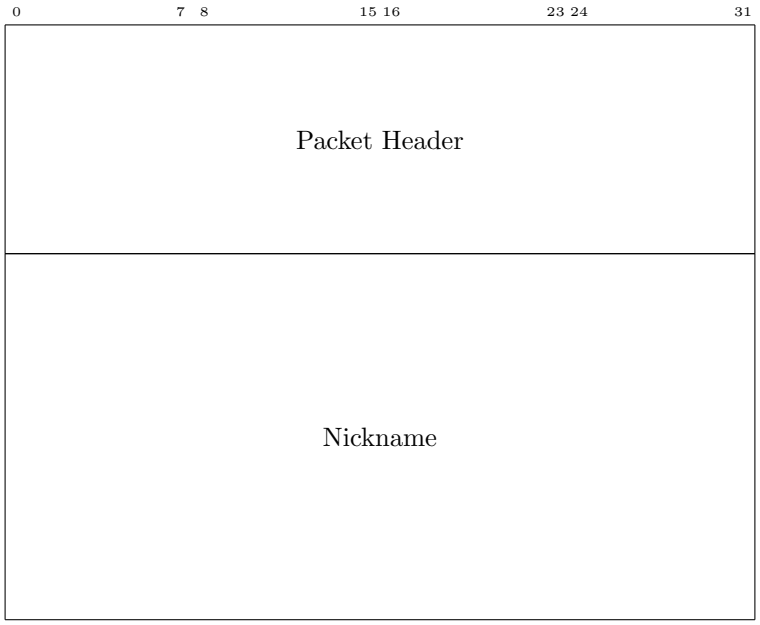

Figure 3: Nickname Packet

3.3.1 Nickname

A 32 character null-terminated string representing the player's nickname.

3.4 Tagged Packet

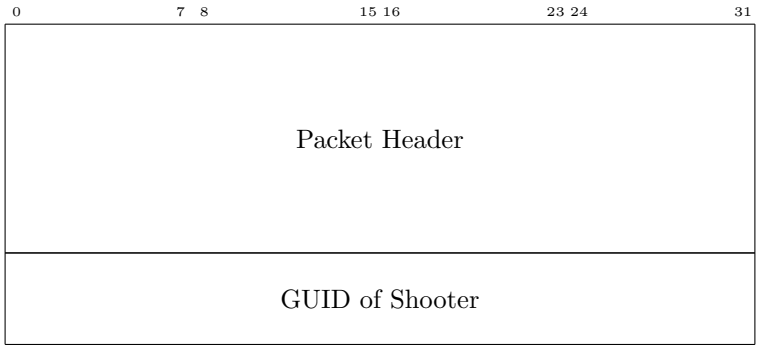

Figure 4: Tagged Packet

3.4.1 GUID of Shooter

The GUID of the client that tagged the local rat.

3.5 Tagged Acknowledgment Packet

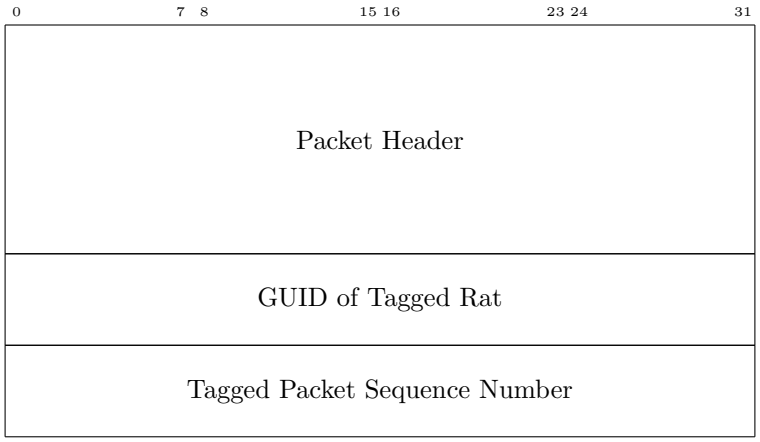

Figure 5: Tagged ACK Packet

3.5.1 GUID of Tagged Rat

The GUID of the client being acknowledged.

3.5.2 Tagged Packet Sequence Number

The sequence number of the packet being acknowledged.

3.6 Leaving Packet

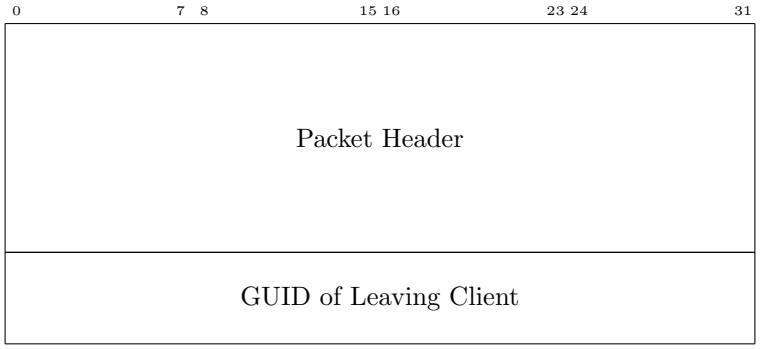

Figure 6: Leaving Packet

3.6.1 GUID of Leaving Client

The GUID of the client that is leaving the game.

3.7 Request for Retransmission Packet

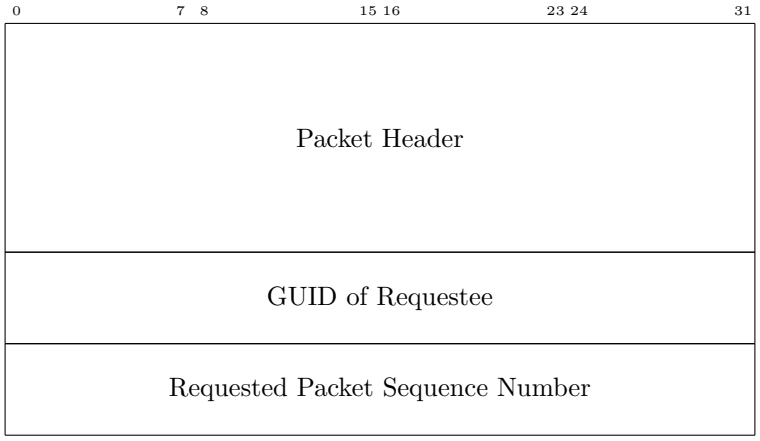

Figure 7: Request for Retransmission Packet

3.7.1 GUID of Requestee

The GUID of the client being asked to retransmit.

3.7.2 Requested Packet Sequence Number

The sequence number of the packet being requested.

4 Timing and Semantics

4.1 Player Moves

The client must send out a *state packet* to all other players for each event causing it to change its local state. Local state includes player position and direction, projectile position and direction, and score. In addition, it must send at least one state packet every 500ms.

4.2 Collisions

A collision is defined as two or more objects attempting to occupy the same cell at the same time. An object may be a rat or a projectile. Two events are defined to occur at the *same time* if their timestamps are within 250ms of each other.

4.3 Player Movement Collision Resolution

If a collision occurs between two rats, the contended tile will be occupied by the rat whose *state packet* contained the lowest CRT. The other rat must revert to its previous position.

4.3.1 Tag Detection

A tag occurs when a projectile collides with a rat. The determination that a tag has occurred is made at the tagged client, and this client is responsible for notifying the shooter via a *tagged packet*. The tagged client must continue to retransmit the *tagged packet* every 500ms until an acknowledgement is received or the shooter has left the game.

4.3.2 Contended Tag Resolution

If there is ambiguity regarding which remote client tagged the local client then the method of awarding the tag is implementation defined, so long as exactly one shooter is selected.

4.4 Joining a Game

A client joining a game discovers the state of existing players during the *discovery phase* as described in Section 2.1. Upon transition to the *active phase* the newly joined client begins transmitting packets. Other clients dynamically discover the new client by receiving its traffic.

4.5 Leaving a Game

A client exiting a game must transmit a *leaving packet* which need not be acknowledged. If no traffic is received from a client for 10 seconds then that client is assumed to be treated by remaining clients as if it had exited.

4.6 Nicknames

The *nickname packet* is used to maintain a mapping of client GUIDs to player nicknames for user-interface purposes. Each client must transmit a *nickname packet* at least once every 5 seconds. An implementation is free to define how a player's nickname is displayed if no mapping yet exists.

4.7 Temporary Loss of Contact

Transient network disruptions may cause clients to time out and be removed from one another's game state. Because remote client state is dynamically discovered from received traffic, the game will converge back to a consistent state when communication is restored.

4.8 Game Time Management

A client joining the game establishes a local time epoch during the discovery phase. This epoch must be derived from the timestamps of received state packets. In this way the local time epoch will be synchronized with the peer game times within the network latency. If no traffic is received during the discovery phase then the client must set its epoch to zero. It is assumed that system clock drift between clients is negligible.

4.9 Possible Inconsistencies

4.9.1 Packet Loss

Packet loss due to a transient network disruption may lead to temporary inconsistencies such as undetected collisions. Such temporary artifacts are acceptable because the game state will converge back to a consistent state when the disruption subsides. This assurance is provided by the idempotency of state packets since they use absolute position values.

4.9.2 Out-of-Order Packet Delivery

Processing state packets out of order could lead to erratic movement of remote objects. Therefore packets received with lower sequence numbers than the most recently processed packet for a given client must be dropped.

4.9.3 Clock Drift

The assumption of negligible clock drift may prove to be unwarranted. Significant drift could lead undetected collisions. The simplest way to handle this problem may be to abort the game if significant time skew is detected. Less disruptive but more complex solutions could correct for drift or identify individual offending clients to evict from the game.

4.9.4 GUID collisions

It is technically possible for two clients to randomly generate the same 64-bit GUID. While exceedingly unlikely if a high quality random number generator is used, such an event would cause unacceptable global inconsistencies in the game state. However, for the sake of simplicity no provision is made for detecting and resolving GUID conflicts.