The Ratpack Protocol Specification v0.1

Ahmed Aljunied and Mansoor Malik

Introduction

Ratpack is a protocol that defines the syntax, semantics, and timing rules for allowing multiple players to play Mazewar on computer nodes interconnected by a network.

In this document the term node refers to an instance of an executable program that implements the Ratpack protocol. Each instance is controlled by the actions of a user. The term user and player are used interchangeably in this document.

This protocol allows users to start, join, and leave games. When a game is in progress, a user controls the motion of a rat as it navigates through a maze. Rats can tag other rats by firing missiles. Points are awarded for tagging other rats. Points are deducted for firing missiles and for being tagged.

The shared state of this game consists of rat positions, missile positions, and player scores. One of the primary goals of this protocol is to maintain sufficient consistency. In other words, the shared state may diverge temporarily but it has to converge in an expedient manner.

The Ratpack protocol is layered on top of UDP. All messages are sent to a single multicast address and port. The multicast address and port that is used in our implementation is 224.1.1.1 and 5016 respectively. The protocol provides an addressing mechanism layered over the UDP protocol. Each player is identified using a unique identifier. There is also an identifier that is used for broadcasting messages to all players. A message consists of a common header followed by a payload. The header and message payloads are defined later in this document.

Since messages are sent over an unreliable packet network, the Ratpack protocol must deal with lost packets, delayed packets, and packets that arrive out-of-order. The protocol defines the semantics and timing rules for dealing with these types of events.

In addition, it is possible for a node to fail because of malfunctioning software or hardware. The protocol defines how this type of failure should be detected and what actions should be taken in response.

Protocol Definition
The following table provides a summary of the messages supported by this protocol. A common header precedes each message. The header and messages are defined in more detail later in this document.

JoinGameRequest	All nodes issue a request to join a game. The first node starts a game if it issues 3 requests and receives no responses.
JoinGameResponse	All nodes participating in a game must respond with this message after receiving a JoinGameRequest message.
LeaveGameRequest	A node sends this message when it leaves a game.
LeaveGameResponse	A node sends a response when it receives a LeaveGameRequest message.
LocationStatus	A node sends this message to report its rat's location, direction, and score.
MissileStatus	A node sends this message when it launches a missile and while a missile is in flight.
TagConfirmationRequest	A node sends this request to another node whose rat it has tagged.
TagConfirmationResponse	A node responds to a TagConfirmationRequest with a response that either accepts or rejects a tag.
ForceStopGame	A node stops if it receives this message. The conditions under which this message is sent are explained later.

MessageHeader

	Magic Number	Version	Message Type	Player ID (Source)	Player ID (Destination)	Sequence Number
Byte	0	4	6	7	11	15
Offset	3	5		10	14	18

Magic Number The magic number is 0xdeadbeef.

Version The version number of the Ratpack protocol. The version number consists of two bytes. The first byte indicates the major version number and the second byte indicates the minor version number. The current version number is 0x0001.

Message Type
0x00 = JoinGameRequest
0x01 = JoinGameResponse
0x02 = LeaveGameRequest
0x03 = LeaveGameResponse
0x04 = LocationStatus
0x05 = MissileStatus
0x06 = TagConfirmationRequest
0x07 = TagConfirmationResponse
0x08 = ForceStopGame

Player ID (Source) The identifier (ID) of the player who is the sender of this message.

Player ID (Destination) The identifier (ID) of the player who is the recipient of this message.

Sequence Number The sequence number of this message.

The magic number must be inserted in all headers. If a packet is received with an incorrect magic number, the packet must be discarded. The magic number is required in case an errant application accidentally sends multicast messages to the address and port being used by the nodes participating in a Mazeware game.

The version number is a reserved field that may be used in the future if more than one protocol version is supported. For now, all nodes must insert the value 0x0001. A node must discard any messages that do not have this version number.

Each header is followed by a message. The Message Type field identifies the type of message that follows. A node must discard any packet that does not range from 0x00 to 0x08.

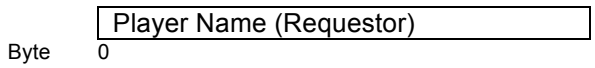
A player ID is a random number chosen by a player when joining a game. The rules for selecting an ID are defined in the JoinGameRequest section later in this document. The value of this field can range from 1 to (2^32)-1 and must be unique. The value of 0x00000000 is reserved as a broadcast address and should not be used.

A node must correctly handle the player IDs when sending and receiving messages. When sending messages, a node should insert the ID of its player in the source field. The destination player ID field can either be an individual node or the broadcast address depending on the type of message.

All messages are multicast to all other players. Hence a node must examine each incoming message and discard any message where the destination player ID does not match its own identifier or the broadcast address. A node should also examine the source player ID and discard any messages that match its own ID. This may happen since messages are multicast to all nodes, including the node that transmitted a message.

The sequence number is a monotonically increasing value that starts at one and increases by one every time a sender transmits a UDP packet. A receiver uses it to detect out-of-order packets from a sender. This allows a receiver to ignore older status updates when new ones have already been received. The maximum value of a sequence number is (2^32)-1. This value is large enough to ensure that a player session ends before sequence numbers wrap around. The behavior is undefined in the unlikely event that sequence numbers wrap around.

JoinGameRequest



Byte Offset 0 31

Player Name (Requestor) The name selected by a user when starting a Mazewar game. The name does not have to be unique and may be blank. The name has a maximum length of 32 characters and must be terminated with a NULL character which has an ASCII value of 0.

A node sends a JoinGameRequest message to join a game that is in progress. Each host must acknowledge the message by responding with a JoinGameResponse message.

The JoinGameRequest message must be sent to the 224.1.1.1 multicast address on port number 5016. The node that is sending this message must select a random identifier for its player. This value must range from 1 to (2^32)-1. This identifier is inserted in the source player ID field in the header. The destination player ID field in the header must be set to 0x00000000 as the message is broadcast to all players.

The request must include the player name of the node that is sending the request. Each player name is up to 32 characters and terminates with a NULL value (ASCII 0). A name does not have to be unique and it may be empty.

After a node transmits this message, it must be able to handle all the scenarios described below.

A node must set a timer for 1 second and wait for a response. If no response is obtained, a node must retransmit 2 additional times. The node will make use of exponential delay and will wait 2 seconds during the 2nd attempt, and 4 seconds during the 3rd attempt. Multiple attempts are necessary in case the network is experiencing temporary congestion that could result in delays or packet loss. If no response is received after the third attempt, the node can assume that it is the first one to start and wait for other nodes to join.

With the exception of the first node, all subsequent nodes should receive a response. The maximum limit of 3 retries only applies if no response is received. A node should give up immediately if it receives a response indicating that the maximum number of players has been reached. It should retry as many times as possible if it receives a response indicating that it selected a duplicate player ID.

If multiple nodes are trying to join a game, a node will get a response with a code of RetryJoinInProgress. If a node retries two additional times and gets a response with a RetryJoinInProgress code, it should give up. This is described in more detail in the JoinGameResponse section. If however, a node does not get a RetryJoinInProgress while retrying then it should continue and join the game using the 1, 2, and 4 seconds time intervals as described earlier.

JoinGameResponse

	Response Code	Player Name (Responder)	Total Players
Byte Offset	0	1 32	33

Response Code
0x00 = Accept
0x01 = RejectMaxPlayerLimitReached
0x02 = RetryDuplicatePlayerID
0x03 = RetryAnotherJoinInProgress

Player Name (Responder) The name of the player sending the response. The name has a maximum length of 32 characters and must be terminated with a NULL character which has an ASCII value of 0.

Total Players The total number of players participating in a game. The maximum number is limited to 8 in the current version of the protocol. This value can be increased to 127 in future versions.

All nodes playing in a game must respond with a JoinGameResponse message after receiving a JoinGameRequest message.

The message must be transmitted to the 224.1.1.1 multicast address on port 5016. The source player ID field in the header must be set to match the ID of the sender. The destination player ID field in the header must be set to ID of the node that sent the JoinGameRequest message.

The responder must examine the player ID of the node that sent the JoinGameRequest. If this value is 0x00000000 or the ID of a node currently playing a game, the response code must be set to 0x02 (RetryDuplicatePlayerID). A node that has already joined a game must take special care to ignore any subsequent JoinGameResponse messages. This can happen, as the destination player ID in the header will match both the existing node and the new node trying to join a game.

The responder must also verify that the maximum player count is not reached. If this occurs, the response code of the JoinGameResponse message must be set to 0x01 (RejectMaxPlayerLimitReached).

The response must include the player name of the node that is sending the response. Each player name is up to 32 characters and terminates with a NULL value (ASCII 0). A name does not have to be unique and it may be empty.

The response must include the total number of players that are currently playing a game. This is necessary in the event of packet loss to or from other nodes. The node that made the request will notice the mismatch between this value and the number of responses that it received. This can occur if either a request or a response was lost or delayed. In either case, the original requestor must resend a JoinGameRequest message. The original requestor can send a maximum of 3 JoinGameRequests. The requestor must

wait for 1, 2, and 4 seconds for responses. If all nodes have not responded by this point, it should quit without sending any other messages.

To ensure a consistent shared state, a node that sends a JoinGameResponse with an Accept response code cannot assume that the node that made the request will actually join. A node should be transitioned from a joining to playing state after it sends its first LocationStatus update. The count of the total number of players playing a game should not be incremented until this occurs. A node should wait a maximum of 7 seconds for a LocationStatus message. If no LocationStatus message is received before this timeout occurs, it should delete any resources created for the node in the joining state.

Multiple nodes may attempt to join a game at once. The following timing rules must be followed to prevent inconsistencies from occurring:

(1) Two nodes may send a JoinGameRequest at the same time. The two new nodes must not only join with the existing nodes playing the game but also with each other. To address this, only one node can join at a time. A node that receives two or more consecutive JoinGameRequest messages should accept the first request and wait for a maximum of 7 seconds for that first node to send a LocationStatus message before accepting any further JoinGameRequest messages. The node should send a response with a RetryAnotherJoinInProgress response code to any other nodes attempting to join a game while it is waiting. A node that gets a message with a RetryAnotherJoinInProgress response code should backoff using random exponential delay and retry 2 additional times before giving up. The wait interval before re-trying on each attempt is [0-2] seconds, [0-4] seconds.

(2) A maximum of 8 players can play in a game. If 7 nodes are currently playing a game, it is possible for two nodes to simultaneously attempt to join a game. One of the requests should be accepted and the other one denied. But the two JoinGameRequest messages may be processed in different order by the 7 nodes that are currently playing games. So the same scheme as described earlier should be used. This scheme will ensure that either one node joins or neither node joins.

LeaveGameRequest

This message consists of a header only and has no payload.

A node that wishes to leave a game sends a LeaveGameRequest message to all other nodes. This message must be sent to the 224.1.1.1 multicast address on port 5016. The source player ID field in the header must be set to match the ID number of the sender. The destination player ID field in the header must be set to 0x00000000 as the message is broadcast to all players.

A node must set a timer for 1 second after sending a LeaveGameRequest message and wait for a response from all other nodes participating in a game. If a response is missing from one or more nodes, a node must retransmit the LeaveGameRequest message 2 additional times. The node will make use of exponential delay and will wait 2 seconds during the 2nd retransmission, and 4 seconds during the 3rd retransmission. Multiple attempts are necessary in case the network is experiencing temporary congestion that could result in delayed or lost packets. A node can leave a game after its third attempt even if it is missing a response from a node.

LeaveGameResponse

This message consists of a header only and has no payload.

A node that receives a LeaveGameRequest message must respond with a LeaveGameResponse message.

The message must be sent to the 224.1.1.1 multicast address on port 5016. The source player ID field in the header must be set to match the ID of the sender. The destination player ID field in the header must be set to ID of the node that sent the LeaveGameRequest message.

A node that sent a LeaveGameRequest message may retransmit two additional times. This may occur if the response from one or more nodes was lost or delayed due to network congestion. Thus a node should set a timer for 7 seconds after the first request is received and be ready to respond to two additional requests before the timer expires. Once a timer has expired, a node should free up all local resources allocated to the player who left the game.

LocationStatus

	X Loc	Y Loc	Direction	Score
Byte	0	2	4	5
Offset	1	3		8

X Loc The X coordinate of the player's rat.

Y Loc The Y coordinate of the player's rat.

Direction The direction the rat is facing.
0x00 = North
0x01 = South
0x02 = East
0x03 = West

Score The player's current score. The score can be a positive or negative value. The values can range from –(2^31) to (2^31)-1.

A node sends a LocationStatus whenever a rat moves, changes direction, or one second has passed since the last time it sent a LocationStatus. A node should use the LocationStatus updates from other players to update the position of all rates in their maze.

The LocationStatus message must be sent to the 224.1.1.1 multicast address on port 5016. The source player ID field in the header must be set to match the ID number of the sender. The destination player ID field in the header must be set to 0x00000000 as it must be broadcast to all players.

This message is also used as a mechanism to detect node failures. If a total of 10 seconds have elapsed since a LocationStatus update was last received from a node then that node is considered to have failed. All resources for that node should be freed up.

The recipient of a LocationStatus message should ensure that the player ID of the sender is valid. If it is not, it should send a ForceStopGame message to the sender.

Two rats cannot occupy the same cell in a maze. Thus, the recipient of a LocationStatus message must verify that another rat does not already occupy the same cell. If this occurs, the recipient must discard the LocationStatus update and take no action. The shared state (rat locations) will temporarily diverge but will converge again when one of the rats moves to an unoccupied cell.

Each node is responsible for keeping track of the score of its player, and its player only. To ensure a consistent shared state, a node should not attempt to increment or decrement the score of other players. It should only increment or decrement its own score and advertise it in the LocationStatus updates that it transmits.

All recipients are responsible for tracking the sequence numbers of LocationStatus updates as packets may be delivered out of order. If a node receives an update with a lower (older) sequence number, it should discard the LocationStatus message.

MissileStatus

	X Loc	Y Loc	Direction	Missile ID
Byte	0	2	4	5
Offset	1	3		8

X Loc The X coordinate of the missile.

Y Loc The Y coordinate of the missile.

Direction The direction the missile is heading.
0x00 = North
0x01 = South
0x02 = East
0x03 = West

Missile ID Each missile launched by a player should have a unique ID.

A node sends a MissileStatus message when a missile is launched. It then sends updates after 0.2 second have elapsed since the last time it sent a MissileStatus message. It stops sending updates when a missile hits another rat or a wall. A player cannot fire a new missile if an existing missile is still in flight.

The MissileStatus message must be sent to the 224.1.1.1 multicast address on port 5016. The source player ID field in the header must be set to match the ID number of the sender. The destination player ID field in the header must be set to 0x00000000 as it is broadcast to all players.

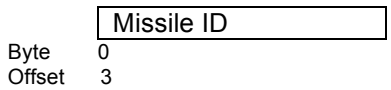
Missiles travel at a rate of one maze cell every 0.2 seconds. Once a node receives the first MissileStatus message, it should advance the missile in its local game without getting further updates. It should discard any subsequent updates that match the player ID and missile ID of the first message. The reason that a sender sends more than one MissileStatus message is because of potential packet losses. If the first message is lost, subsequent messages can nonetheless reach other players.

When a player fires a missile, it should reduce its score by one. The updated score will be transmitted to the other player when the next LocationStatus message is sent.

The ID of the first missile should be 0. The ID of each subsequent missile should be increased by one. The missile ID can therefore range from 0 to (2^32)-1. The maximum value is large enough to ensure that a game will end before this value wraps around. The behavior is undefined in the unlikely event that this value does wrap around.

Because of timing differences, a rat may be tagged in one game but escape in another game. Both players must agree on a rat being tagged before scores can be incremented by the shooter or deducted by the player whose rat was tagged. The TagConfirmationRequest and TagConfirmationResponse messages are used for that purpose and described in the next section.

TagConfirmationRequest



Missile ID Each missile launched by a player should have a unique ID.

A node sends a TagConfirmationRequest when it has tagged a rat in its local copy of the game. A node cannot claim points for tagging a rat until it has received a TagConfirmationResponse from the node that it tagged.

The TagConfirmationRequest message must be sent to the 224.1.1.1 multicast address on port 5016. The source player ID field in the header must be set to match the ID number of the player whose rat fired a missile. The destination player ID field in the header must match the ID number of the player whose rat was tagged.

A node that receives a TagConfirmationRequest must examine the destination player ID field in the header. It should discard message that are addressed to other nodes.

The missile ID must match the ID of the missile in the MissileStatus message.

A node will wait 1 second for a response. If no response is obtained, a node will retransmit 2 additional times. The node will make use of exponential delay and will wait 2 seconds during the 2nd attempt, and 4 seconds during the 3rd attempt. This is necessary in case the network is experiencing congestion that could result in delays or packet loss. If a player does not receive a response after 3 attempts, it will give up. It will not get any points for tagging a rat.

TagConfirmationResponse

	Missile ID	Tag Code
Byte Offset	0 3	4

Missile ID The missile ID that was in the TagConfirmationRequest message.

Tag Code 0x00 = AcceptTag
0x01 = RejectTag

A node sends a TagConfirmationResponse after it receives a TagConfirmationRequest.

The TagConfirmationResponse message must be sent to the 224.1.1.1 multicast address on port 5016. The source player ID field in the header must be set to match the ID number of the sender. The destination player ID field in the header must match the ID number of the player that requested confirmation of a tag.

The TagConfirmationReponse is used to reach consensus between a player whose rat fired a missile and a player whose rat was tagged. A rat must have been tagged in both instances of a game in order to reach consensus on the score. Each player must track the location of all missiles and advance each missile by one cell for every 0.2 seconds that elapses. Thus, each node is independently responsible for determining if a missile hit a rat.

A node that receives a TagConfirmationRequest must examine the destination player ID field in the header. It should discard message that are addressed to other nodes.

If a node receives a TagConfirmationRequest message for a missile that is still in flight, it should wait until the missile hits a rat or a wall before responding with a response. If a missile hit a players' rat, it must respond with a code of AcceptTag. Otherwise, the code should be RejectTag.

If a node sends a response with a code of AcceptTag it should decrement its own score by 5 and pick a new random position on the maze that is not occupied by another player's rat. It should then immediately send out a new LocationStatus update.

A node that receives a TagConfirmationResponse with an AcceptTag code should increase its score by 11 points. It should not modify its score if the tag code is RejectTag.

It is possible for a TagConfirmationResponse message to be lost in the event of network congestion. In this case, the player that issued the original request may retry two additional times. Hence, a node that responded to the original request must be ready to respond to two subsequent requests. It should set a timer for 7 seconds after receiving the first request and be ready to respond to two additional requests before the timer expires.

If all three TagConfirmationResponse messages are lost then the player who fired the missile will not be able to claim any points. This is acceptable as the shared state of the game (player locations and scores) will be consistent even though a player did not receive points that they were entitled to.

ForceStopGame

This message has no payload. A ForceStopGame message should be transmitted if a message (other than JoinGameRequest) is received from a node with an unknown player ID. This situation may occur if there was a network failure that disconnected one or more nodes. If this occurred then multiple games may be running independently. One mechanism for dealing with this situation is to force nodes to quit.

The ForceStopGame message must be sent to the 224.1.1.1 multicast address on port 5016. The source player ID field in the header must be set to match the ID number of the sender. The destination player ID field in the header must match the ID number of the unknown player.

When a node receives a ForceStopGame message, it must exit immediately. A user who wishes to continue must then restart their game.

Example 1: Player 1 Starts a Game
Player 1 starts and its node picks a random player ID of 0x0a230391. It issues a JoinGameRequest message. The destination address of the UDP datagram is 244.1.1.1 on port 5016. The payload of the UDP datagram is as follows:

	Magic Number	Version	Message Type	Player ID (Source)	Player ID (Destination)	Sequence Number	Player Name
	0xdeadbeef	0x01	0x00	0x0a230391	0x00000000	0x00000001	Player1\0
Byte	0	4	6	7	11	15	19
Offset	3	5		10	14	18	50

No other player is on the network. Hence, no response will be received after 1 second. The node for player 1 will retransmit the request and wait for 2 seconds. It will increment the sequence number in its header for this message and will continue to do so in every subsequent message. Once this timeout expires, the node will retransmit again and wait for 4 seconds. Once this timeout expires, this node will conclude that it is the first node on the network. It will send a LocationStatus message every time its rat position changes or every second, whichever occurs first. The payload of the LocationStatus message is as follows:

Magic Number	Ver.	Msg. Type	Player ID (Src.)	Player ID (Dest.)	Seq. Num.	X	Y	Dir	Score
0xdeadbeef	0x01	0x04	0x0a230391	0x00000000	0x00000003	0x01	0x0a	0x00	0x00000000
Byte 0	4	6	7	11	15	19	21	23	24
Offset 3	5		10	14	18	20	22		27

Example 2: Player 2 Joins a Game
We assume that player 1 has already started as described in the previous example and that player 2 then joins.

The node for player 2 picks a random player ID of 0x000203af. It sends a JoinGameRequest message with a destination address of 224.1.1.1 on port 5016. The payload of the UDP datagram is as follows:

	Magic Number	Version	Message Type	Player ID (Source)	Player ID (Destination)	Sequence Number	Player Name
	0xdeadbeef	0x01	0x00	0x000203af	0x00000000	0x00000001	Player2\0
Byte	0	4	6	7	11	15	19
Offset	3	5		10	14	18	50

The node for player 1 receives this request and responds with a JoinGameResponse message. The destination of the message is 224.1.1.1 on port 5016. The payload of the UDP datagram is as follows:

Magic Number	Ver.	Msg. Type	Player ID (Source)	Player ID (Destination)	Sequence Number	Player Name	Total Players
0xdeadbeef	0x01	0x01	0x0a230391	0x000203af	0x0000001b	Player1\0	1
Byte 0	4	6	7	11	15	19	51
Offset 3	5		10	14	18	50	

The node for player 2 will then pick a random maze cell for its rat that is unoccupied. It will then send a LocationStatus update, which is broadcast to all players. This message is shown below.

Magic Number	Ver.	Msg. Type	Player ID (Src.)	Player ID (Dest.)	Seq. Num.	X	Y	Dir	Score
0xdeadbeef	0x01	0x04	0x000203af	0x00000000	0x00000002	0x08	0x07	0x01	0x00000000
Byte 0	4	6	7	11	15	19	21	23	24
Offset 3	5		10	14	18	20	22		27

When the node for player 1 receives the first LocationStatus message, it will transition player 2 from a joining state to a playing state. Both players will set timers for 10 seconds for the other node. The timer will be reset whenever a LocationStatus update is received. If the timer expires for a player, then the node for that player will be assumed to have failed and the resources for that player will be freed up.

Example 3: Player 1 Fires a Missile

We assume that Player 1 and 2 have already joined a game as described earlier. Player 1 fires a missile to tag the rat of player 2. The following MissileStatus message will be sent when a missile is first fired. Player 1 will advance the missile by one maze cell every 0.2 seconds in its local game. It will send out an update every 0.2 seconds.

Magic Number	Ver.	Msg. Type	Player ID (Src.)	Player ID (Dest.)	Seq. Num.	X	Y	Dir	Missile ID
0xdeadbeef	0x01	0x05	0x0a230391	0x00000000	0x000000a3	0x03	0x03	0x02	0x00000000
Byte 0	4	6	7	11	15	19	21	23	24
Offset 3	5		10	14	18	20	22		27

Player 1 will deduct one point after firing a missile. It will broadcast its new score when it sends its next LocationStatus update.

Once player 2 receives the MissileStatus update from player 1, it will create a missile in its local game and advance it by one maze cell every 0.2 seconds. Player 2 will ignore any subsequent MissileStatus updates from player 1 that match this missile ID.

Example 4: Player 1 and 2 Achieve Consensus on a Rat Being Tagged

We assume that a missile fired by the rat of player 1 hits the rat of player 2. We further assume that this event occurs in the nodes of both player 1 and 2. Player 1 sends a TagConfirmationRequest message to player 2. This message is shown below:

Magic Number	Ver.	Msg. Type	Player ID (Src.)	Player ID (Dest.)	Seq. Num.	Missile ID
0xdeadbeef	0x01	0x06	0x0a230391	0x000203af	0x000000ac	0x00000000
Byte 0	4	6	7	11	15	19
Offset 3	5		10	14	18	22

Player 2 receives the TagConfirmationRequest from player 1 and responds by sending a TagConfirmationResponse message to player 1. This message is shown below.

Magic Number	Ver.	Msg. Type	Player ID (Src.)	Player ID (Dest.)	Seq. Num.	Missile ID	Tag Code
0xdeadbeef	0x01	0x07	0x000203af	0x0a230391	0x0000000f	0x00000000	0x00
Byte 0	4	6	7	11	15	19	23
Offset 3	5		10	14	18	22	

Player 2 will decrement its score by 5 after sending the TagConfirmationResponse message. It selects a new random position for its rat that is unoccupied by another rat. It broadcasts its new location and new score when it sends its next LocationStatus update.

Player 1 will increment its score by 11 after receiving the TagConfirmationRequest from player 2. It broadcasts its new score when it sends out its next LocationStatus update.