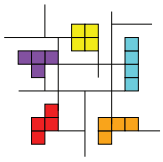
Parisian Master of Research in Computer Science

Network programming project

MindTris Protocol Specification



DGMT Protocol

Revision 1.2.0.4

Authors:
Charles-Pierre Astolfi
Raphaël Bonaque
David Montoya
Émile Contal
Martin Gleize

Supervisors:
Thomas Chatain
Hedi Benzina

January 12, 2011

1 Naming conventions

In this paper, packet structures for the DGMT protocol will be described in detail. We will employ the following types of terms:

- *data_type*: the following data types will be employed:
  - *void*: a void pointer. It can have take only value, `NULL`. This will be used to refer to non assigned variables.
  - *byte*: an ordered collection of 8 bits.
  - *int*: an integer of unspecified precision. We will employ both the decimal `[0-9]+` and hexadecimal `0x[0-9A-F]+` notations in this paper.
    Implentors's note: implementors are free to choose how they store integer data, as long as the precision employed is enough to represent the whole set of expectable values.
  - *int*8, *int*16, *int*32, *int*64: integers whose range is in comprised in $[\![0, 2^x - 1]\!]$, where x is equal to the number that appears in the name.
  - *bool*: a boolean, whose value can be either 1 or 0.
  - *ustring*: a finite sequence of characters from the Unicode character set.
  - *data_type*[]: an array of the specified *data_type*.
- DATA_ENCODING: a byte encoding of a given *data_type*.
- `CONSTANT_VALUE`: a constant byte value, used to distinguish different message types or answers.
- `variable`: a variable used to store sensitive protocol data.

1.1 Encodings

We define the following data encodings:

- BYTE: data is taken as "is", a *byte* array is formed.
- INTEGER: a big-endian integer encoding of an *int*. The actual interpretation depends on the number of bytes employed.
- BOOLEAN: an encoding for a *bool*. Any non zero value represents 1, while a zero value stands for 0. If more than 1 byte is used, a *bool* array is formed.
- STRING: a ASCII encoding for *ustring*, using an arbitrary number of bytes.
- USTRING: a UTF-8 encoding for *ustring*, using an arbitrary number of bytes.
- ARRAY: an encoding for an array of arbitrary data. Individual data has particular encoding, and this data is concatenated at the byte level to form an array. The size of an ARRAY usually refers to the number of elements present in it.

Implementor's note: STRING can actually be interpreted as a USTRING without any loss of data. The reason we define the former is so that we can specify the kind of information we expect to have, although this should be done at a higher level of abstraction.

2 Overview

The DGMT protocol defines how a MindTris client can communicate with a MindTris server.

2.1 Client's state information

Clients must maintain the following state information for each server it connects to. We will describe the state information by defining a series of variables. By default, variables can be of two *data_type*, one of which must be *void*.

Implementor's note: This information does not specify the implementation one should use, it merely gives information relevant to the validity of the protocol. For this reason, some information, which could be important for providing good user experience, will not necessarily be described here.

- `server_pub_key` (*byte*[]): The server's public RSA key. Used by the client in communications where secret transmission of data is necessary. This variable should include the modulus and exponent integer values of the key. The default value is `NULL`.

The client keeps track of the current status of the client/server connection. This could be whether the client has logged in, is in a game, etc. It should include information such as following:

- `connected` (*bool*): Whether or not the client has received a positive handshake answer from the server. Defaults as 0.
- `logged_on` (*bool*): Whether or not the client has successfully authenticated itself as a certain user existing in the server's database. Defaults as 0.
- `user` (*ustring*): The user the client has authenticated as. Defaults as `NULL`.
- `lobby_id` (*int*32): The ID of the lobby the user has joined. Defaults as `NULL`.
- `peer_id` (*int*8): The peer ID for the lobby the user is in. Defaults as `NULL`.
- `creator_lobby_id` (*int*32): The ID of the lobby the user has created, if it has created any lobby. Defaults as `NULL`.
- `session_id` (*int*64): the id of the session the server has generated for the current game. Defaults as `NULL`.
- `am_playing`(*bool*): Whether or not the user is in a game. Defaults as 0.

Conversely, server must maintain state information from each client it's connected to. This should include `connected`, `logged_on`, `ser`, `lobby_id`, `peer_id`, `creator_lobby_id` and`am_playing`. Additionally, it will include the client's public key (`client_pub_key`), which will be used for communications between peers to authenticate their messages. This key defaults as `NULL`.

The server should at all times maintain information about lobbies that have been created, which will include their lobby ID, session ID and password. It should also keep a database of all registered users.

2.2 Public Key message structure

In some parts of the protocol, public RSA keys are expected to be transmitted. The expected format for such transmission is the following:

Name	Encoding	Size (B)	Default	Comment
Modulus Size	INTEGER	2	N/A	The size of the Modulus field.
Modulus	INTEGER	varies	N/A	The modulus part of the public key.
Exponent Size	INTEGER	1	N/A	The size of the Exponent field.
Exponent	INTEGER	varies	N/A	The modulus part of the public key.

We will refer to this as the DGMT RSA Public Key format.

In other parts, a public DSA key is expected. This is the expected format:

Name	Encoding	Size (B)	Default	Comment
DSA_p Size	INTEGER	2	N/A	The size of the DSA_p field.
DSA_p	INTEGER	varies	N/A	The modulus part of the public key.
DSA_q Size	INTEGER	2	N/A	The size of the DSA_q field.
DSA_q	INTEGER	varies	N/A	The modulus part of the public key.
DSA_g Size	INTEGER	2	N/A	The size of the DSA_g field.
DSA_g	INTEGER	varies	N/A	The modulus part of the public key.
DSA_y Size	INTEGER	2	N/A	The size of the DSA_y field.
DSA_y	INTEGER	varies	N/A	The modulus part of the public key.

3 Client-Server messages

All packets sent between the server and the client must be structured as follows:

Name	Encoding	Size (B)	Default	Comment
Protocol	STRING	4	"DGMT"	Default protocol identifier.
Size	INTEGER	2		The size of the message in bytes, including the header.
Payload	BYTE	varies		The content of this field will vary depending on the type of message.

3.1 Hello

This message is the first message in the handshake between the server and the client. The client will first send a `HELLO_FROM_CLIENT` message to the server, who in turn should answer with a `HELLO_FROM_SERVER` message.

Request:

Name	Encoding	Size (B)	Default	Comment
Type	INTEGER	1	0x00	This identifies the message as `HELLO_FROM_CLIENT`.
Protocol Version	INTEGER	4	1.2.0.4	Protocol/Client version; the format is xx.xx.xx.xx.

Response:

Name	Encoding	Size (B)	Default	Comment
Type	INTEGER	1	0x80	This identifies the message as `HELLO_FROM_SERVER`.
Server Answer	INTEGER	1	N/A	Server's response. Must be one of the following: 0x00 (`SUCCESS`), 0x01 (`WRONG_PROTOCOL_VERSION`), 0x02 (`UNKNOWN_ERROR`).
Server's Public Key	BYTE	varies	N/A	The server's public RSA key. Used for transmission of secret data from the client to the server. The expected format is the DGMT RSA Public Key format.
Message Size	INTEGER	2	N/A	The size of the Message field (bytes)
Message	USTRING	varies	N/A	A human readable message from the server to the client. This could be a welcome message, if the Server Answer was `SUCCESS`, or a few details about the error that occurred.

If the server answers `SUCCESS`, the client will store the server's public key in its `server_pub_key` variable. Additionally the state variable `connected` is set to 1. Any other messages exchanged between the server and the client assume that `connected` is 1.

3.2 Keep alive

Clients regularly send a message with an empty Payload to the server from the moment they received a `HELLO_FROM_SERVER`. They are expected to do so every 60 seconds, otherwise the server will drop the connection, and terminate any ongoing transactions with the client.

3.3 User creation

By this process, a client can create a user to be stored in the server's database. The client will first send a `CREATE_USER` message to the server, who in turn should answer with a `USER_CREATION` message. This message is only valid when the client `connected` variable is set to 1.

Request:

Name	Encoding	Size (B)	Default	Comment
Type	INTEGER	1	0x01	This identifies the message as `CREATE_USER`.
Username Size	INTEGER	1	N/A	The size of the Username field.
Username	USTRING	varies	N/A	A name used to login.
Display Name Size	INTEGER	1	N/A	The size of the Display Name field.
Display Name	USTRING	varies	N/A	A name that will be displayed to other users.
Email Size	INTEGER	2	N/A	The size of the Email field. This value cannot be greater than 320.
Email	STRING	varies	N/A	An email address, used to recover a lost password.
Encrypted Password Size	INTEGER	2	N/A	The size of the Encrypted Password field.
Encrypted Password	BYTE	varies	N/A	We will describe this content below.

The Encrypted Password field should contain the following data, encrypted with the RSAES-OAEP (RSA Encryption Scheme - Optimal Asymmetric Encryption Padding) (SHA-1) scheme, using `server_pub_key`.

Name	Encoding	Size (B)	Default	Comment
Password	STRING	varies	N/A	The password that will be used to login.

Valid usernames must match `[:alpha:][:alnum:_.-]*`, and valid password must match

```
^.*(?=.{6,})(?=.*[:alnum:])(?=.*[:punct:]).*$
```

Response:

Name	Encoding	Size (B)	Default	Comment
Type	INTEGER	1	0x81	This identifies the message as `USER_CREATION`.
Answer	BYTE	1	N/A	• 0x00: the user has been created with success. • 0x01: this username already exists. • 0x02: invalid username. • 0x03: invalid password. • 0x04: invalid email.

3.4 Login

By this process, a client can login as a user present in the server's database. The client will first send a `LOGIN` message to the server, who in turn should answer with a `LOGIN_REPLY` message. This message is only valid when the client `connected` variable is set to 1.

Request:

Name	Encoding	Size (B)	Default	Comment
Type	INTEGER	1	0x02	This identifies the message as `LOGIN`.
Username Size	INTEGER	1	N/A	The size of the Username field.
Username	USTRING	varies	N/A	A name used to login.
Encrypted Password Size	INTEGER	2	N/A	The size of the Encrypted Password field.
Encrypted Password	BYTE	varies	N/A	We will describe this content below.

The Encrypted User Login Info field should contain the following data, encrypted with the RSAES-OAEP (SHA-1) scheme, using `server_pub_key`.

Name	Encoding	Size (B)	Default	Comment
Password	STRING	varies	N/A	The password that will be used to login.

Response:

Name	Encoding	Size (B)	Default	Comment
Type	INTEGER	1	0x82	This identifies the message as `LOGIN_REPLY`.
Answer	BYTE	1	N/A	• 0x00: login success. • 0x01: username does not exist. • 0x02: bad username/password. • 0x03: too many tries, try again later. • 0x04: login success, but another instance was disconnected elsewhere.
Answer Payload	BYTE	varies	N/A	The content of this field is determined by the Answer field.

When Answer is equal to 0x00, the Answer Payload field will contain the data structure below. It should be empty otherwise.

Name	Encoding	Size (B)	Default	Comment
Display Name Size	INTEGER	1	N/A	The size of the Display Name field.
Display Name	USTRING	varies	N/A	The user's name that is displayed to others.

Normally, the server will only give a success response if the username and password provided by the client match in the server's database. The variable `logged_on` is set to 1 and `user` is set to the provided username on both the server and the client's side.

3.5 Lobby Creation

By this process, a user can create a lobby. This is only possible if `logged_on` is set to 1, `lobby_id` is set to `NULL` and `creator_lobby_id` is set to `NULL`.

Request:

Name	Encoding	Size (B)	Default	Comment
Type	INTEGER	1	0x03	This identifies the message as `CREATE_LOBBY`.
Lobby Name Size	INTEGER	1	N/A	The size of the Lobby Name field.
Lobby Name	USTRING	varies	N/A	The name of the lobby to be created.
Player Allowed Count	INTEGER	1	N/A	The maximum number of players allowed.
Has Password	BOOLEAN	1	0	Whether or not the lobby will require a password.
Encrypted Lobby Password Size	INTEGER	2	N/A	The size of the Encrypted Lobby Password field or empty if the Has Password field is 0.
Encrypted Lobby Password	BYTE	varies	N/A	This field is empty if the Has Password field is 0.
Port number	INTEGER	2	N/A	The client's TCP listen port number, used for communication between peers.
Client's Public Key	BYTE	varies	N/A	The client's public key, used to sign peer-to-peer messages. The expected format is the DGMT Public Key format.

The Encrypted Lobby Password field, if provided, should contain the following data, encrypted with the RSA-OAEP (SHA-1) scheme, using `server_pub_key`.

Name	Encoding	Size (B)	Default	Comment
Password	STRING	varies	N/A	The password that will requested to join the lobby.

Note that a valid password must match

```
^.*(?=.{6,})(?=.*[:alnum:])(?=.*[:punct:]).*$
```

Response:

Name	Encoding	Size (B)	Default	Comment
Type	INTEGER	1	0x83	This identifies the message as `LOBBY_CREATION`.
Answer	BYTE	1	N/A	• 0x00: lobby created with success. • 0x01: invalid password. • 0x02: you do not have enough rights create a lobby. • 0x03: invalid number of players. • 0xFF: unknown error.
Lobby Creation Payload	BYTE	varies	N/A	The value of this field depends on the Answer field. This is described below.

When Answer is equal to 0x00, the following information is included in the Lobby Creation Payload field:

Name	Encoding	Size (B)	Default	Comment
Lobby ID	INTEGER	4	N/A	An ID generated by the server for the lobby than has been created. This value should be generated so that all current available lobbies have unique IDs.
Peer ID	INTEGER	1	N/A	The creator peer's unique ID for this lobby. Used to identify peers during the game.
Session ID	INTEGER	8	N/A	This lobby's session's ID. Used in peer-to-peer messages to prevent clients from reusing signed packets. This is different from lobby ID in the sense that a Session ID is required to be unpredictable, while lobby IDs can be generated for efficient retrieval of lobby lists.

The field is empty otherwise.

3.6 Lobby List Retrieval

Users might want to know the list of available lobbies. This is only possible if `logged_on` is set to 1.

Request:

Name	Encoding	Size (B)	Default	Comment
Type	INTEGER	1	0x04	This identifies the message as `GET_LOBBY_LIST`.

Response:

Name	Encoding	Size (B)	Default	Comment
Type	INTEGER	1	0x84	This identifies the message as `LOBBY_LIST`.
Lobby List Size	INTEGER	1	N/A	The size of the Lobby List array (number of elements).
Lobby List	ARRAY	varies	N/A	An array of Lobby data, described below.

The Lobby List contains an array with the following data structure:

Name	Encoding	Size (B)	Default	Comment
Lobby ID	INTEGER	4	N/A	The lobby ID.
Lobby Name Size	INTEGER	1	N/A	The size of the Lobby Name field.
Lobby Name	USTRING	varies	N/A	The name of the lobby to be created.
Player Count	INTEGER	1	N/A	The number of players present in the lobby.
Player Allowed Count	INTEGER	1	N/A	The maximum number of players allowed.
Password Protected	BOOLEAN	1	0	Whether or not this lobby requires a password to join.
Creator Size	INTEGER	1	N/A	The size of the Creator field.
Creator	USTRING	varies	N/A	The display name of the creator of this lobby.

3.7 Joining a lobby

Users can decide to join a particular lobby. This is only possible if `logged_on` is set to 1 and if `lobby_id` is `NULL`.

Request:

Name	Encoding	Size (B)	Default	Comment
Type	INTEGER	1	0x05	This identifies the message as `JOIN_LOBBY`.
Lobby ID	INTEGER	4	N/A	The lobby ID.
Password Size	INTEGER	1	N/A	The size of the Password field.
Password	STRING	varies	N/A	The lobby's password.
IP Address	INTEGER	4	N/A	The client's IPv4 address, as described in RFC 791.
Port number	INTEGER	2	N/A	The client's TCP listen port number, used for communication between peers.
Client's Public Key	BYTE	varies	N/A	The client's public key, used to sign peer-to-peer messages. The expected format is the DGMT DSA Public Key format.

Response:

Name	Encoding	Size (B)	Default	Comment
Type	INTEGER	1	0x85	This identifies the message as `JOINED_LOBBY`.
Lobby ID	INTEGER	4	N/A	The lobby ID.
Answer	BYTE	1	N/A	• 0x00: joined lobby with success. • 0x01: wrong password • 0x02: lobby is full • 0x03: unknown error
Answer Payload	BYTE	varies	N/A	The content of this field is determined by the Answer field.

When Answer is equal to 0x00, the Answer Payload field will contain the data structure below. It should be empty otherwise.

Name	Encoding	Size (B)	Default	Comment
Lobby Name Size	INTEGER	1	N/A	The size of the Lobby Name field.
Lobby Name	USTRING	varies	N/A	The name of the lobby to be created.
Player Allowed Count	INTEGER	1	N/A	The maximum number of players allowed.
Creator Peer ID	INTEGER	1	N/A	The Peer ID of the creator of this lobby.
Peer ID	INTEGER	1	N/A	The joining peer's unique ID for this lobby. Used to identify peers during the game.
Session ID	INTEGER	8	N/A	This lobby's session's ID. Used in peer-to-peer messages to prevent clients from reusing signed packets. This is different from lobby ID in the sense that a Session ID is required to be unpredictable, while lobby IDs can be generated for efficient retrieval of lobby lists.
Client List Size	INTEGER	1	N/A	The size of the Client List array (number of elements).
Client List	ARRAY	varies	N/A	An array of clients currently present in the lobby, described below.

The Client List array should implement the following data structure:

Name	Encoding	Size (B)	Default	Comment
Peer ID	INTEGER	1	N/A	The peer's unique ID for this lobby. Used to identify peers during the game.
Display Name Size	INTEGER	1	N/A	The size of the Display Name field.
Display Name	USTRING	varies	N/A	The name used by the client.
IP Address	INTEGER	4	N/A	The client's IPv4 address, as described in RFC 791.
Port number	INTEGER	2	N/A	The client's TCP listen port number, used for communication between peers.
Client's Public Key	BYTE	varies	N/A	The client's public key, used to sign peer-to-peer messages. The expected format is the DGMT DSA Public Key format.

Once the server responds, both ends set the `lobby_id` variable for this client to the value given by

3.8 Leaving a Lobby

A user can decide to leave a lobby. In this case, it sends the following message to the server:

Name	Encoding	Size (B)	Default	Comment
Type	INTEGER	1	0x06	This identifies the message as `LEAVE_LOBBY`.

Note that this message will only be valid if `logged_on` is set to 1 and if `lobby_id` is different than `NULL`. The server, instead of responding directly to this message, will respond with an `UPDATE_CLIENT_STATUS` message, which is described in section 3.10.

3.9 Kicking a user

The user who created the lobby can decide to kick another user. This message has to be sent:

Name	Encoding	Size (B)	Default	Comment
Type	INTEGER	1	0x07	This identifies the message as `KICK_USER_FROM_LOBBY`.
Peer ID	INTEGER	1	N/A	The ID of the peer who will be kicked.

Note that this message will only be valid if `logged_on` is set to 1, `lobby_id` is different than `NULL` and if `creator` is set to 1. The server, instead of responding directly to this message, will respond with an `UPDATE_CLIENT_STATUS` message, which is described in section 3.10.

3.10 Update client lobby status

The server is supposed to notify clients in a lobby of any status updates from other clients joining, leaving or being kicked from the lobby. The server sends this message to all present clients (that is, those whose `lobby_id` matches the lobby in question), including the one whose status has been updated, if they happen to have been kicked or left the lobby.

Name	Encoding	Size (B)	Default	Comment
Type	INTEGER	1	0x88	This identifies the message as `UPDATE_CLIENT_STATUS`.
Lobby ID	INTEGER	4	N/A	The ID of the lobby in question.
Status Update	BYTE	1	N/A	• 0x00: has joined the lobby • 0x01: has left the lobby • 0x02: has been kicked from the lobby
Peer ID	INTEGER	1	N/A	The ID of the peer whose status has been updated.
Status Payload	BYTE	varies	N/A	Depending on the type of Status Update, the content of this field will differ.

When Status Update is equal to 0x00, the following information is included in the Status Payload field:

Name	Encoding	Size (B)	Default	Comment
Display Name Size	INTEGER	1	N/A	The size of the Display Name field.
Display Name	USTRING	varies	N/A	The name used by the joining client.
IP Address	INTEGER	4	N/A	The joining client's IPv4 address, as described in RFC 791.
Port number	INTEGER	2	N/A	The joining client's TCP listen port number, used for communication between peers.
Client's Public Key	BYTE	varies	N/A	The client's public key, used to sign peer-to-peer messages. The expected format is the DGMT DSA Public Key format.

When the Status Update is equal to 0x01 or 0x02, clients are expected to terminate the connection they have with the corresponding peer. Additionally, the variable `lobby_id` is set to `NULL` and `creator` is set to 0.

3.11 Starting a game

Users having created a lobby can choose to start a game. They are easily identified for having `creator_lobby_id` different than `NULL`.

Request:

Name	Encoding	Size (B)	Default	Comment
Type	INTEGER	1	0x10	This identifies the message as `START_GAME`.

Response:

Name	Encoding	Size (B)	Default	Comment
Type	INTEGER	1	0x90	This identifies the message as `GAME_STARTING`.
Answer	BYTE	1	N/A	• 0x00: engaging with game start procedures.

The server's answer is only interesting when an unknown error is triggered. Otherwise, a correct game start procedure will force the server into telling every client in the lobby matching the client's `creator_lobby_id` to start loading a game, a message we will describe in the next section.

3.12 Loading a game

The server can request clients having joined a certain lobby to start loading a game. Games can take a certain amount of time to allocate resources in a given client, which is why we expect clients to take some time before they give an answer to the server. Additionally, server includes first pieces to be played during the game.
Request:

Name	Encoding	Size (B)	Default	Comment
Type	INTEGER	1	0x91	This identifies the message as `LOAD_GAME`.
First Pieces Array Size	INTEGER	1	10	The size of the First Pieces array.
First Pieces	ARRAY	varies	N/A	An array including the IDs of the first pieces being played, each of them is encoded as 1-byte INTEGER.

Note: Pieces are encoded from 0 to 6, as follows:

Piece Name	Piece ID
I	0
J	1
L	2
O	3
S	4
T	5
Z	6

Response:

Name	Encoding	Size (B)	Default	Comment
Type	INTEGER	1	0x11	This identifies the message as `LOADED_GAME`.
Answer	BYTE	1	N/A	• 0x00: loaded game, connected to all peers • 0x01: can't connect to certain peers
Answer Payload	BYTE	varies	N/A	The content of this field is determined by the Answer field.

When Answer is equal to 0x01, the Answer Payload field will contain the data structure below. It should be empty otherwise.

Name	Encoding	Size (B)	Default	Comment
Peer ID Array Size	INTEGER	1	N/A	The size of the peer ID array.
Peer IDs	ARRAY	varies	N/A	The IDs of each of peer this peer couldn't connect to. Each of these is represented as a 1-byte INTEGER.

3.13 Beginning of a game

The server, once it received a successful `LOADED_GAME` answer from all connected players, will decide as to the beginning of a game. It will simply send this message to all of them:

Name	Encoding	Size (B)	Default	Comment
Type	INTEGER	1	0x92	This identifies the message as `BEGIN_GAME`.

3.14 Obtaining new pieces

The server is responsible for telling each player about the next pieces to be played. It can do so any time it wishes. For this purpose, this message is sent to each client:

Name	Encoding	Size (B)	Default	Comment
Type	INTEGER	1	0x93	This identifies the message as `NEW_PIECES`.
Piece Number Offset	INTEGER	4	N/A	The number of the first piece in the Pieces array.
Pieces Array Size	INTEGER	1	10	The size of the Pieces array.
Pieces	ARRAY	varies	N/A	An array including the IDs of the new pieces, each of them is encoded as 1-byte INTEGER.

Additionally, clients can encourage the server to do so:

Name	Encoding	Size (B)	Default	Comment
Type	INTEGER	1	0x13	This identifies the message as `GIVE_NEW_PIECES`.
Piece Number Offset	INTEGER	4	N/A	The number of the first piece requested.
Piece Requested Number	INTEGER	1	N/A	The number of pieces requested.

Reaction by the server to this message is done at the server's discretion.

3.15 End of a game

Clients send the results of a game to the server when it finishes:

Name	Encoding	Size (B)	Default	Comment
Type	INTEGER	1	0x14	This identifies the message as `GAME_END`.
Results Array Size	INTEGER	1	N/A	The size of the Results array.
Results	ARRAY	varies	N/A	An array of results for each different client.

The Results array will include the following:

Name	Encoding	Size (B)	Default	Comment
Peer ID	INTEGER	1	N/A	The ID of the peer whose result is from.
Winning Rank	INTEGER	1	N/A	Ranks determine the order at which players lost. The player who ultimately won should be rank 1. Players whose rank can't be determined should be rank 0.
Score	INTEGER	4	N/A	The score of the client at the end of the game.

4 Peer-to-Peer messages

Peer-to-peer messages work inherently different from Server/Client messages. Unlike the former, peer-to-peer do not follow a request-response pattern.

All packets sent between the peers must be structured as follows:

Name	Encoding	Size (B)	Default	Comment
Protocol	STRING	7	"DGMTP2P"	Default protocol identifier.
Size	INTEGER	2		The size of the message in bytes, including the header.
Payload	BYTE	varies		The content of this field will vary depending on the type of message.

Basically, clients are expected to connect to each other the moment they join a lobby. The client having joined last must initiate the connection. They must use the TCP port each individual peer is listening to, as provided by the server. Before they engage in any conversation, they must go through a handshake phase.

4.1 Handshake

Before engaging in any real communication, peers must authenticate with each other, by following a secure three-way handshake procedure.

During this phase, one peer initiates the connection. This is done by sending a `CONNECTION_REQUEST` message, to which the other peer will respond with a `CONNECTION_ACCEPTED`

message, if the request is valid. The handshake ends with the peer's initiating sending a `CONNECTION_ACKNOWLEDGED` message, acknowledging the other peer's authenticity, and responding to the other's peer challenge.

Connection request

Name	Encoding	Size (B)	Default	Comment
Type	INTEGER	1	0x00	This identifies the message as `CONNECTION_REQUEST`
Lobby ID	INTEGER	4	N/A	The lobby ID the peers are connected to.
Initiating Peer ID	INTEGER	1	N/A	The ID of the peer who initiates the connection.
Listening Peer ID	INTEGER	1	N/A	The ID of the peer that accepts the connection.
Initiating Challenge Code	INTEGER	8	N/A	A random number issued by the peer initiating the connection, for the purpose of only one handshake attempt.

The peer accepting the connection, will check whether the Lobby ID, Initiating Peer ID and Listening Peer ID are valid. If they are not, he should terminate the TCP connection. Otherwise he will send the following message:

Connection challenge accepted

Name	Encoding	Size (B)	Default	Comment
Type	INTEGER	1	0x01	This identifies the message as `CONNECTION_ACCEPTED`
Lobby ID	INTEGER	4	N/A	The lobby ID the peers are connected to.
Initiating Peer ID	INTEGER	1	N/A	The ID of the peer who initiates the connection.
Listening Peer ID	INTEGER	1	N/A	The ID of the peer that accepts the connection
Initiating Challenge Code	INTEGER	8	N/A	The challenge code issued by the peer initiating the connection.
Listening Challenge Code	INTEGER	8	N/A	A random number issued by the peer accepting the connection, for the purpose of only one handshake attempt.
Signature Size	INTEGER	2	N/A	The size of the signature field in bytes.
Signature	BYTE	varies	N/A	The accepting peer's signature for the message between the Lobby ID and Listening Challenge Code. The signature is generated by the Digital Signature Algorithm.

The peer initiating the connection will check if the message corresponds to an ongoing handshake request, and verify that the signature matches the one it has for the peer accepting the connection, as provided by the server. If the message is not valid, he should terminate the TCP connection. Otherwise, the peer will assume that he has initiated a connection with the proper peer, and will send the final message:

Connection Challenge Acknowledged

Name	Encoding	Size (B)	Default	Comment
Type	INTEGER	1	0x02	This identifies the message as `CONNECTION_ACKNOWLEDGED`
Lobby ID	INTEGER	4	N/A	The lobby ID the peers are connected to.
Initiating Peer ID	INTEGER	1	N/A	The ID of the peer who initiates the connection.
Listening Peer ID	INTEGER	1	N/A	The ID of the peer that accepts the connection
Initiating Challenge Code	INTEGER	8	N/A	The challenge code issue by the peer initiating the connection.
Listening Challenge Code	INTEGER	8	N/A	The challenge code issued by the peer accepting the connection.
Signature Size	INTEGER	2	N/A	The size of the signature field in bytes.
Signature	BYTE	varies	N/A	The initiating peer's signature for the message between the Lobby ID and Listening Challenge Code. The signature is generated by the Digital Signature Algorithm.

The accepting peer will verify that the fields match a previous `CONNECTION_ACCEPTED`, and then check the signature for the peer initiating the connection, as provided by the server. If OK, he assumes that a connection has been established with the proper peer. Otherwise, he should terminate the TCP connection.

4.2 Chat

Chat messages don't rely on the server for transmission. They are sent directly to each peer.

Chat Message:

Name	Encoding	Size (B)	Default	Comment
Type	INTEGER	1	0x10	This identifies the message as `CHAT_SEND`.
Session ID	INTEGER	8	N/A	The lobby's session's ID. This makes reusing of old signed packets harder.
Chat Message Length	INTEGER	2	N/A	The size of the Chat Message field.
Chat Message	USTRING	varies	N/A	A chat message sent by the peer we are connected to.
Signature Size	INTEGER	2	N/A	The size of the signature field in bytes.
Signature	BYTE	varies	N/A	The client's RSA signature for the Session ID and Chat message, generated with DSA.

4.3 Game Round packets

Most importantly, peers are supposed to send each other packets for each round that has been played. Rounds happen every 100ms.

Name	Encoding	Size (B)	Default	Comment
Type	INTEGER	1	0x11	This identifies the message as `ROUND`.
Round Data	BYTE	varies	N/A	The data about the round, this is described below.
Signature Size	INTEGER	2	N/A	The size of the signature field in bytes.
Signature	BYTE	varies	N/A	The client's RSA signature for the Round Data field, generated with DSA.

Round Data include a bunch of gameplay dependent data.

Name	Encoding	Size (B)	Default	Comment
Session ID	INTEGER	8	N/A	The lobby's session's ID. This makes reusing of old signed packets harder.
Round Number	INTEGER	4	N/A	The round number. Games start at round number 0, and this number increases by 1 for each subsequent round.
Moves Array Size	INTEGER	1	N/A	The size of the Moves array.
Moves	ARRAY	varies	N/A	An array of moves the player has made during the round.
Round Data Hashes Array Size	INTEGER	1	N/A	The size of the Round Data Hashes field.
Round Data Hashes	ARRAY	varies	N/A	An array of peer hashes, this is described next.

The Tetris board is a rectangle of 10 columns by 20 rows. The X and Y offset range between 0-9, and 0-19, respectively. The location of a piece is given by the X and Y offset of the bottom left corner of the rectangle that perfectly bounds the piece, given its current orientation. The Moves array will include the following:

Name	Encoding	Size (B)	Default	Comment
Piece Number	INTEGER	4	N/A	The number of the piece being moved. Games start with piece number 0, and this number increases by 1 for each new piece that comes.
Piece Orientation	INTEGER	1	N/A	The orientation of the piece. The default piece position is 0x00, and this number increases by 1 each time the piece is rotated in a clockwise fashion.
Piece X Offset	INTEGER	1	N/A	The X offset of the location where the piece has been dropped.
Piece Y Offset	INTEGER	1	N/A	The Y offset of the location where the piece has been dropped.

The Round Data Hashes array should include the following information for each player present in the game.

Name	Encoding	Size (B)	Default	Comment
Peer ID	INTEGER	1	N/A	The Peer ID of the peer whose hash is from.
Round Data Hash	BYTE	20	N/A	The SHA-1 hash of the Round Data field sent by the above client.

4.4 Keep alive

Peers regularly send each other a message with an empty Payload to the server from the moment they established a connection. They are expected to do so every 60 seconds.

4.5 Relaying

It is possible for peers connected to the same lobby to relay messages between each other. This can be useful when the peer-to-peer network is a ring or a tree. For this purpose, they use the following message structure:
From Message:

Name	Encoding	Size (B)	Default	Comment
Type	INTEGER	1	0x12	This identifies the message as `MESSAGE_FROM`.
From Peer ID	INTEGER	1	N/A	The ID of the peer whose the message comes from.
Message Length	INTEGER	2	N/A	The size of the Message field.
Message	BYTE	varies	N/A	A DGMTP2P message, which includes the header.