**Status of this Memo**

This memo defines an experimental game protocol to be used by BACKGAMMON clients (Player Client) and servers (Application Server).

**Abstract**

BACKGAMMON protocol is a text-based protocol that is to be built on top of the Transmission Control Protocol (TCP). The protocol has client-to-server.

The protocol features Unicode-encoded protocol messages that are text based and are framed as TCP segments. The header of the protocol message defines the type of the message, and every message has a number of expected responses.

**Revision history**

|  |  |  |
| --- | --- | --- |
| Version | Changed By | Date |
| 1 | Initial Draft | 21.12.2014 |
| 2 | Changed Game notation. Modified THROWDICE, SENDMOVE response messages. Added PCWATCH command. Removed PCHBEAT message and Added SRVHBEAT message. | 31.12.2014 |
|  |  |  |

* INTRODUCTION

The BACKGAMMON protocol is inspired by the Hypertext Transmission Protocol (HTTP, RFC2616) and Internet Relay Chat Standards (IRC, mainly RFC1459). The protocol adheres to the following general rules:

* The transmission between endpoints of the protocol complies with the request-response paradigm. That is, all traffic exchanged between two endpoints will be initiated (request) by one of the endpoints and will either be acknowledged or answered (response) by the other endpoint
* The BACKGAMMON protocol is Half Duplex, that is, a requesting client is blocked until a fitting response is received. This hints that a queueing mechanism will need to be set up for incoming messages.
* Only clients of the BACKGAMMON applications send out Requests. Application Servers are designed as passive elements that issue only responses.
* Application servers are stateless. That is, although they keep and maintain the state information of the clients, they do not have multiple states that necessitate a different execution path for the same incoming request.
* Player clients are state-full, in terms of the BACKGAMMON application. That is, they may have different states which call for different paths of execution such as being “Waiting for play”, “Playing”, “Watching a game”, etc.
* To ensure request to response matching on both sender and receiver, a sequence id "username" field shall be appended to bodies of each BACKGAMMON message. A BACKGAMMON message without a "username" field is invalid.
* In almost all cases, IP addresses are not explicitly communicated between elements of the system. It is both the client and the server's responsibility to extract, store and manage IP addresses of other system elements they are in connect with. i.e. It is the responsibility of the server to figure out which client sent a certain message by working out the IP address of the message and matching it to an internal IP table
* As a backgammon notation, Paul Magriel notation is used. <http://en.wikipedia.org/wiki/Backgammon_notation>
* MESSAGE FORMAT

2.1. The BACKGAMMON message format is standard across PC-AS protocols.

2.3. BACKGAMMON protocol messages adhere to the following format:

<HEADER>

<BODY>

2.3.1 **Message Header** The<HEADER> portion of the message is a 8-character identifier that defines the content and aim of the message. All characters of the header must be ASCII characters. Legal forms of the HEADER are presented below:

**PC-AS Client Requests**

* PCCONN: Client connect to server
* PCREQPLAY: Client sends request for play
* PCREQWATCH: Client sends request for watching game
* PCWATCH: Client starts watching game
* PCTHROWDICE: Client sends throw dice request
* PCSENDMOVE: Client sends move
* PCWRONGMOVEALERT: Client sends wrong move alert
* PCBEAROFF: Client removes stone from the board
* PCEND: Client ends connection
* PCOK: Client Response OK

**PC-AS Server Responses**

* SRVOK: Server Response OK
* SRVERR: Server Error
* SRVHBEAT: Server Hear-beat
* SRVACKDICE: Server acknowledges dice result to players
* SRVACKMOVE: Server acknowledges move to players.
* SRVACKWRONGMOVE: Server acknowledges wrong move alert to opponents

2.3.2 **Message** each message contains a username (username). The username is alphanumeric and sent by the requester It should be unique name within concurrent users. It is used to identify matching request-response pairs on the server side. That is, the same "username" in the request is echoed to the requester in the response message as well.

CCONN ()

{

“username” : “sebnema”,

“ip” : “10.10.12.55”

}

2.4. BACKGAMMON Protocol messages shall be transmitted to the corresponding client/server's TCP port 9897 as BACKGAMMON/PC-AS protocol.

2.5. BACKGAMMON Protocol is a strictly request-response protocol. A TCP connection shall be **opened and closed** for each request-response pair.

* PC - AS Protocol

The Player Client – Application Server protocol (BACKGAMMON/PC -AS Protocol) is a protocol for player clients to connect to a server and communicate throughout the game.

The PC-AS protocol is strictly request-response, with clients able to issue requests, and servers sending one and only one response message for each request to all clients. The servers of the BACKGAMMON system can issue requests to clients.

Requests and responses of the PC-AS protocol are presented below.

* PC - AS Requests
* PCCONN : Client Connect

PCCONN is the client request that is used to connect a client to the server. BACKGAMMON clients issue the connection request once a login dialog has been directed to the server.

The server is then expected to reply with SRVOK (Server Response OK), and return a welcome message; or a SRVERR message explaining why the user was denied access.

Request Header: PCCONN

Request Fields:

* username: unicode string, unique username used for connection

Expected Responses: SRVOK, SRVERR

Example Request:

PCCONN

{

"username" : “sebnema”,

“ip” : “10.10.1.22”

“serverip” : “10.24.56.78”

“serverport” : 9898

}

Example Response:

SRVOK

{

"message" : "Hi sebnema, You are connected to 10.24.56.78,9898"

}

SRVERR

{

"message" : "Username sebnema already exists. Choose another name"

}

* PCREQPLAY: Client request for play

PCREQPLAY is that the player client sends to server to request for start playing. After connect BACKGAMMON clients request for play and waits for server match an opponent to it.

The server is expected to reply either with SRVOK or SRVERR.

If SRVOK returns and if request list returns an opponent name. The response message will be generated as “Your opponent is {opponent user name}”and displayed on client screen.

Request Header: PCREQPLAY Request Fields:

* username: A valid username

Expected Responses: SRVOK, SRVERR

Example Request:

PCREQPLAY

{

"username": "sebnema"

}

Example Response:

SRVOK

{

"message" : "Successfull."

"opponent" : "foouser",

"gameid" : 1222

}

SRVERR

{

"message" : "No active user to play. Wait or return back for other options"

}

* PCREQWATCH : Client sends request for watching game

PCREQWATCH is used for clients wanting to send watching a game. Since these player clients will be either not assigned to a game or engaged in another game, PCREQWATCH requests are sent to the server.

The server is expected to reply either with SRVOK or SRVERR. If the server detects that if there are any active game session, then server shall choose a game randomly and respond with a SRVOK. If there is not any active game session, SRVERR response shall return.

Request Header: PCREQWATCH Request Fields:

* username: string, username for requesting user

Example Request:

PCREQWATCH

{

"username": “username”

}

Example Response:

SRVOK

{

"message" : "Successful."

"gameid": 123123

}

SRVERR

{

"message": "No active game session to watch. Wait or return back for other options."

}

* PCPLAY : Client starts playing game

PCPLAY is used for clients to start playing a game.

The server is expected to reply either with SRVOK or SRVERR. If the server detects that if there are any active game session, then server shall return stone-table respond with a SRVOK. If there is not any active game session (Game is finished or failed), SRVERR response shall return.

Request Header: PCPLAY Request Fields:

* username: string, username for requesting user
* gameid: unique identifier of active game session return from server as a result of watching request.

Example Request:

PCPLAY

{

"username": “username”,

"gameid": 123123

}

Example Response:

SRVOK

{

“gameid” : 123123,

"board":

{

"whites": [ {6(1)}, {8(3)}, {24(1)}, {13(2)}, {7(4)} ],

"blacks": [ {20(2)}, {19(5)}, {17(3)}, {14(1)}, {4(1)} , {12(3)} ]

}

}

SRVERR

{

“gameid” : 123123

"message": "Error occured"

}

* PCWATCH : Client starts watching game

PCWATCH is used for clients to start watching a game.

The server is expected to reply either with SRVOK or SRVERR. If the server detects that if there are any active game session, then server shall return stone-table respond with a SRVOK. If there is not any active game session (Game is finished or failed), SRVERR response shall return.

Request Header: PCWATCH Request Fields:

* username: string, username for requesting user
* gameid: unique identifier of active game session return from server as a result of watching request.

Example Request:

PCWATCH

{

"username": “username”,

"gameid": 123123

}

Example Response:

SRVOK

{

“gameid” : 123123,

"board":

{

"whites": [ {6(1)}, {8(3)}, {24(1)}, {13(2)}, {7(4)} ],

"blacks": [ {20(2)}, {19(5)}, {17(3)}, {14(1)}, {4(1)} , {12(3)} ]

}

}

SRVERR

{

"message": "No active game session found. Wait or return back for other options."

}

* PCTHROWDICE: Client sends throw dice request

PCTHROWDICE notifies the server that the requesting client throws dice.

Request Header: PCTHROWDICE

Request Fields:

* username.
* gameid

Expected Responses: SRVOK, SRVERR

Example Request:

PCTHROWDICE

{

"username": “sebnema”,

"gameid": 213123

}

Example Response:

SRVOK

{

“gameid” : 213123

“dice”: “42”

}

SRVERR

{

"message": "Error occured while getting dice."

}

SRVERR

{

"message": "Lost connection to opponent user."

}

* PCSENDMOVE: Client sends move

PCSENDMOVE notifies the server that player client send stone move.

Request Header: PCSENDMOVE

Request Fields:

* username.
* gameid
* move

Expected Responses: SRVOK, SRVERR

Once player client sent PCSENDMOVE, in response, the server is expected to send back a SRVOK response to users.

Moreover, the server shall send SRVMOVEACK to player and watcher clients to acknowledge about the move. In BACKGAMMON/PC-AS protocol, the server acknowledges the player clients once it receives the request.

Example Request:

PCSENDMOVE

{

"username": “sebnema”

“gameid” : “12312”

"move": “42: 8/4 6/4”

}

Player shall send his/her username and id of the game. Additionally, the detail of the dice roll and the move in move object.

Dice rolls are shown as "4-2", denoting a roll of four on one die and two on the other. Moves are recorded using the notation:

4-2:8/4 6/4

This denotes a roll of 4-2, and the corresponding checker moves from point 8 to 4 and from 6 to 4.

If player rolls doubles, plays the numbers shown on the dice twice (E.g. 1-1, 3-3, 6-6), then move object shall include 4 moves.

E.g. 22: 6/4(3) 13/11 means. The first 3 stones move from point 6 to 4 and forth stone moves from point 13 to 11.

In the case of doubles, when all four numbers cannot be played, the player must play as many numbers as he can. Thus, move object shall include less than 4 moves.

When only one number can be played, the player must play that number. Then, move object shall include only 1 move.

Or if either number can be played but not both, the player shall play one. Then move object shall include only 1 move.

When neither number can be used, the player loses his turn (“Gele”). Then move object shall not include any item.

Finally, it is common to use the words "bar" and "off" to describe moves where stones are entered from the bar or taken off during bear off.

For instance bar/22 17/9, and 5/off 2/off.

Example Response:

SRVOK

{

"username": “sebnema”,

"message": “Moved”

}

* PCWRONGMOVEALERT: Client sends wrong move alert

In the BACKGAMMON system, rules of notifying player about wrong move are as follows:

* If user sends his move and opponent doesn’t agree with the move, then opponent might send wrong move alert only right after player’s move.
* Opponent cannot send wrong move alert after s/he threw dice or moved his/her stone.
* User cannot send wrong move alert, right after opponent threw dice.

Request Header: PCWRONGMOVEALERT

Request Fields:

* username, the username of the requested (receiving) user
* gameid

Expected Responses: SRVOK (a brief acknowledgement verifying that the server has received the request), SRVERR acknowledging users if the wrong move alert is not appropriate.

Example Request:

PCWRONGMOVEALERT

{

"username": “sebnema”,

"gameid": 1123123

}

Example Response

SRVOK

{

"username": “sebnema”

“gameid” : 1231233

“message” : “Move is legal”

}

SRVERR

{

"username": “sebnema”

“gameid” : 1231233

“message” : “Move is not legal”

"previousGameboard":

{

"whites": [ {6(1)}, {8(3)}, {24(1)}, {13(2)}, {7(4)} ],

"blacks": [ {20(2)}, {19(5)}, {17(3)}, {14(1)}, {4(1)} , {12(3)} ]

}

}

After receiving this request, the server acknowledges players with the previous stone table position. The server sends SRVWRONGMOVEACK to players with stone table data.

* PCBEAROFF: Client removes stone from the board

In the BACKGAMMON system, rules of bearing off a stone are as follows:

* If user has moved all of his fifteen checkers into his home board, then the user can send bear off request.
* A player can remove stone from his/her own board and with his/her color.
* User cannot send bear off request if there is not a stone at the point that corresponds to dice.
* If there is not a legal move, then user shall not remove any stone.

Request Header: PCBEAROFF

Request Fields:

* username, the username of the requested (receiving) user
* gameid
* move

5-6 means dice rolls. 5/off means stone removed from point 5. Same is for 6/off.

Expected Responses: SRVOK (a brief acknowledgement verifying that the server has received the request), SRVERR acknowledging users if the wrong move alert is not appropriate.

Example Request:

PCBEAROFF {

"username": “sebnema”,

"gameid": 1123123,

“move” : “5-6: 5/off 6/off”

}

Example Response

SRVOK

{

"username": “sebnema”

“gameid” : 1231233

“message” : “successfully removed”

"gameboard":

{

"whites": [ {6(1)}, {8(3)}, {24(1)}, {13(2)}, {7(4)} ],

"blacks": [ {20(2)}, {19(5)}, {17(3)}, {14(1)}, {4(1)} , {12(3)} ]

}

}

After receiving this request, the server returns stone table to acknowledge players about latest stone table.

* PCEND: Client ends connection

PCENDC has taken place if any one of the players or watchers in a game session may request to end the session. This is done by transmitting a PCENDC message to the clients. An End request may be initiated by a client by explicitly clicking an "End Connection" button or closing out of the client window in the client GUI, and shall be explicitly reported to the opponent and watcher clients (i.e. "Opponent ended connection", etc.)

After this request has been acknowledged, the game session is ended, and a new game session may only be formed by another PCREQPLAY or PCREQWATCH message.

If a player client unexpectedly exit the system, the opponent player is initially unaware of the situation. It shall only be aware when no acknowledgement to its PCTHROWDICE, PCSENDMOVE, PCBEAROFF messages are received. In this case, the player shall also end game session, making itself available to other game requests.

Request Header: PCEND

Request Fields:

* username: required, string.

Expected Responses: SRVOK,SRVERR

Example Request:

PCEND

{

"username": “sebnema”

}

* PCOK : Client Response OK

Signifies that the requested operation has been successful. The response may include a set of different response fields according to which BACKGAMMON/PC-AS request it is responding to. The standard response and associated fields are listed explicitly below.

Response Fields:

* username: required. The id of the user sent by the requesting client in the request body
* message: optional
* PC - AS Responses

As also depicted some of them above, the application server issues responses, namely SRVOK and SRVERR, SRVDICEACK, SRVMOVEACK, SRVWRONGMOVEACK.SRVASSIGNOPP: Server assigns an opponent

* SRVOK : Server Response OK

Signifies that the requested operation has been successful. The response may include a set of different response fields according to which BACKGAMMON/PC-AS request it is responding to. The standard response and associated fields are listed explicitly below.

Response Fields:

* username: required. The id of the user sent by the requesting client in the request body
* message: optional
* SRVERR : Server Error

The SRVERR response is sent to the client when the request could not be fulfilled due to a user error or a server error. The reason for the error may optionally be included in the "message" field.

Response Fields:

* username: required. The id of the user sent by the requesting client in the request body
* message: optional, may be used to describe the error
* SRVHBEAT : Server Heart-beat

A server heartbeat is a "dummy" message issued by application server to client check the client is still connected, and expects traffic.

BACKGAMMON application server is expected to issue SRVHBEAT messages regularly at an interval of 30 seconds.

BACKGAMMON application server is expected to consider a client offline, if no heartbeat message is received in the last 30 seconds. For each SRVHBEAT message, the client is expected to respond with either a PCOK (Player Client OK) or PCERR (Player Client Error).

Request Header: SRVHBEAT

Request Fields: username,

Expected Responses: PCOK

**Warning**: While a server sends hear-beat message to client, requests in waiting list shall be sent to the client at every SRVHBEAT, then client response (SRVOK) in order to confirm matched opponent player

Example Request:

SRVHBEAT

{

"username": “sebnema”

"Opponent":

{

"username": "foouser",

"date": "2014-12-21T18:00:00"

}

}

Example Response:

PCROK

{

"username": “sebnema”,

"message": “Connected”

}

In the heartbeat message request, it is also possible to send opponent details which is matched randomly to the user (if user is in waiting list). Opponent object keeps that opponent details.

* SRVACKDICE: Server acknowledges dice results to players

SRVDICEACK is an acknowledgement request from server to player and watcher clients to display dice numbers on the screen.

The server shall send SRVDICEACK message to all clients after player sent PCTHROWDICE message.

Request Header: SRVDICEACK

Request Fields:

* gameid: required, int.
* player: username of the player threw dice
* dice: required, array.

Expected Responses: No response expected

Example Request:

SRVDICEACK

{

"gameid": 123123,

“player” : “sebnema”

“dice”: “4-2”

}

* SRVACKMOVE Server acknowledges move to players

Upon receiving PCSENDMOVE message, server acknowledging a SRVACKMOVE Request. The player and watcher clients receiving the request updates the stone table on the screen.

Request Header: SRVACKMOVE

Request Fields:

* gameid: required, int.
* player: username of the player threw dice
* gameboard: required, array.

Expected Responses: no response expected

Example Requests:

SRVACKMOVE

{

"gameid": 123123,

“player” : “sebnema”

"gameboard":

{

"whites": [ {6(1)}, {8(3)}, {24(1)}, {13(2)}, {7(4)} ],

"blacks": [ {20(2)}, {19(5)}, {17(3)}, {14(1)}, {4(1)} , {12(3)} ]

}

}

* SRVACKWRONGMOVE: Server acknowledges wrong move alert to opponents

After a player client send PCWRONGMVEALERT, the server shall send SRVACKWRONGMOVE message to all player and watcher clients in order to acknowledge about wrong move.

Request Fields:

* username, the username of the requested (receiving) user
* gameid,
* previousgameboard: required, array

Expected Responses: SRVOK (a brief acknowledgement verifying that the server has received the request), SRVERR acknowledging users if the wrong move alert is not appropriate.

Example Request:

SRVACKWRONGMOVE

{

"username": “sebnema”,

"gameid": 1123123,

"previousgameboard":

{

"whites": [ {6(1)}, {8(3)}, {24(1)}, {13(2)}, {7(4)} ],

"blacks": [ {20(2)}, {19(5)}, {17(3)}, {14(1)}, {4(1)} , {12(3)} ]

} }

Expected Responses: no response expected