Backgammon BC-BS Protocol

Status of This Memo

This memo defines an experimental backgammon gaming protocol to be used by Backgammon clients (BC) and servers (BS) for the Internet.

Abstract

BC-BS protocol is a text-based protocol that is to be built on top of the Transmission Control Protocol (TCP). The protocol has one channel: between client and server. It mostly follows request-response style but in some situations there are deviations.

The protocol features ASCII-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. In addition to the header, the body of the message is defined as a Javascript Object Notation (JSON) compliant message that effectively transmits the message content.

Table of Contents

1. INTRODUCTION

The BC-BS protocol is inspired by SWE544 lectures and author's own imagination. The protocol adheres to the following general rules:

- The transmission between clients and server mostly follows request-response style but not always. Server may send different messages to the clients if conditions are held and may not expect any answer from the client.
- Not only clients but also server may send requests.to the clients.
- The protocol folows client / server architecture.
- 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.

2. MESSAGE FORMAT

2.1. The BC-BS message format is standard across BC-BS protocol.

2.2. BC-BS protocol messages implement ASCII encoding. This goes for the message header and body.

2.3. BC-BS protocol messages adhere to the following format:

<HEADER>
[CRLF]
<BODY>

2.3.1. **Message Header** The <HEADER> portion of the message is a 6-character identifier that defines the content and aim of the message. The header portion is also encoded in ASCII. Legal forms of the HEADER are presented below:

BC-BS Client Requests

- CLOGIN: Client Login
- CREQST: Client Request
- CTDICE: Client Throw Dice
- CMOVEC: Client Move Checker

BC-BS Server Responses

- SLRSPS: Server Login Response
- SREQRP: Server Request Response
- STDICE: Server Thrown Dice
- SRJCTM: Server Reject Move

BC-BS Server Requests

- SVPING: Server Ping
- SMOVEC: Server Move Checker

BC-BS Client Responses

- CLPONG: Client Pong
- CTDICE: Client Throw Dice
- CRJCTM: Client Reject Move

BC-BS Server Messages

- SSTOVR: Server Set Over
- SGMOVR: Server Game Over
- STEARD: Server Teardown
- SVRNOK: Server Not Ok

2.3.2. **Message Body** The <HEADER> and <BODY> portions of the message are separated with a blank line (line with nothing but a CRLF). The <BODY> section of the message is to comply with Javascript Object Notation (JSON, defined in http://www.json.org/), and the required and optional fields that will be postulated for each message type shall be defined as "key-value" pairs within the JSON structure. For example, the following message shows an example of a Backgammon / BC-BS protocol message adhering to 2.3.1 – 2

```
CLOGIN
(CRLF)
{
        "userid": "foobar",
}
```

2.4. BC-BS protocol messages shall be transmitted to the corresponding client/server's TCP port 10001. Server should listen to TCP port 10001 for incoming traffic, client should connect to TCP

port 10001 for outgoing traffic.

2.5. A TCP connection shall be opened in the begining and maintained till client leaves the session or a player wins the game or connection is lost for some reason.

3. BC-BS PROTOCOL

3.1. BC-BS Client Requests

CLOGIN: Client Login

CLOGIN is the client request that is used to log in to the server. Backgammon clients issue the login request when they want to connect to the server.

The server is expected to reply with SLRSPS (Server Login Response) showing either a result of success and a welcome message or a result of failure and a message explaining the failure.

Request Header: CLOGIN

Request Fields:

- userid: username used for login

Expected Responses: SLRSPS

Example Request:

CLOGIN

```
{
        "userid": "foobar",
}
```

Example Responses:

SLRSPS

```
{
        "result": "success",
        "msg": "Hi foobar",
        "msg": "You are connected to server_IP:10001",
        "msg": "I want to play",
        "msg": "I want to watch",
}
```

SLRSPS

```
{
        "result": "fail",
        "msg": "foobar is already exists",
        "msg": "Choose another user name and try to reconnect again",
        "msg": "Connection is closed",
}
```

CREQST: Client Request

CREQST is the client request that is used to play or watch a game or leave the server.

- play request is issued when the client wants to play a game
- watch request is issued when the client wants to watch a game
- leave request is issued when the client wants to leave the server

The server is expected to reply with SREQRP (Server Request Response).

Request Header: CREQST

Request Fields:

- type: either play or watch or leave

Expected Responses: SREQRP

Example Requests:

CREQST

```
{
        "type": "play",
}
```

CREQST

```
{
        "type": "watch",
}
```

CREQST

```
{
        "type": "leave",
}
```

Example Responses:

Success response for **play** request. A user *foo* playing with a user *bar*

message to user *foo*

SREQRP

```
{
        "type": "play",
        "result": "success",
        "opponent": "bar",
        "color": "white",
        "turn": 1,
        "msg": "Your turn. Start playing"
```

}

message to user *bar*

SREQRP

```
{
	"type": "play",
	"result": "success",
	"opponent": "foo",
	"color": "black",
	"turn": 0,
	"msg": "Not your turn. Wait your opponent's move"
}
```

Fail response to **play** request.

SREQRP

```
{
	"type": "play",
	"result": "fail",
	"msg": "you are put to waiting list",
}
```

Success response to **watch** request.

SREQRP

```
{
	"type": "watch",
	"result": "success",
	"boardstate": "in backgammon notation",
}
```

Fail response to **watch** request.

SREQRP

```
{
	"type": "watch",
	"result": "fail",
	"msg": "No match to watch",
	"msg": "I want to play",
	"msg": "I want to watch",
}
```

Success response to **leave** request.

SREQRP

```
{
```

```
	"type": "leave",
	"result": "success",
}
```

CTDICE: Client Throw Dice

CTDICE is the client request that is used to instruct the server to throw dice. This request is issued by the client when a new set or game begins. Apart from the initial state, Backgammon clients issue the throw dice request when their turn comes but in this case it purpose is twofold: Accept the move made by opponent so can be viewed as a response and a request from server to throw dice.

The server is expected to reply with STDICE (Server Thrown Dice) in backgammon notation.

Request Header: CTDICE

Expected Responses: STDICE

Example Request:

CTDICE

```
{
}
```

Example Response:

STDICE

```
{
	"dice": "in backgammon notation",
}
```

CMOVEC: Client Move Checker

CMOVEC is the active client's (whose turn to play) request that is used to submit a move. Backgammon clients issue the move request after dice is thrown and it's their turn to play.

The server is expected to reply with either STDICE (Server Thrown Dice) or SRJCTM (Server Reject Move) depending on passive client's (waiting opponent's move) response.

Request Header: CMOVEC

Request Fields:

- move: move in backgammon notation

Expected Responses: STDICE or SRJCTM

Example Request:

CMOVEC

```
{
```

```
        "move": "in backgammon notation",
}
```

Example Responses:

STDICE

```
{
        "dice": "in backgammon notation",
}
```

SRJCTM

```
{
        "boardstate": "in backgammon notation",
}
```

3.2. BC-BS Server Responses

SLRSPS: Server Login Response

SLRSPS is the server response to the client login request (CLOGIN).

The server is expected to reply with SLRSPS (Server Login Response) when confronted with a client login connection request. It shows the success or failure of the connection request.

Response Header: SLRSPS

Response Fields:

- result: success or fail
- msg: any message string sent by the server

Example Responses:

SLRSPS

```
{
        "result": "success",
        "msg": "Hi foobar",
        "msg": "You are connected to server_IP:10001",
        "msg": "I want to play",
        "msg": "I want to watch",
}
```

SLRSPS

```
{
        "result": "fail",
        "msg": "foobar is already exists",
        "msg": "Choose another user name and try to reconnect again",
        "msg": "Connection is closed",
}
```

SREQRP: Server Request Response

SREQRP is the server response to the client request (CREQST).

The server is expected to reply with SREQRP (Server Request Response) when confronted with a client demanding to play or watch a game or leave the server.

Response Header: SREQRP

Response Fields:

- type: play or watch or leave
- result: success or fail
- msg: any message string sent by the server
- opponent: opponent's username
- color: checker's color. black or white.
- turn: 1 is player's turn, 0 is opponent's turn
- boardstate: "in backgammon notation"

Example Responses:

Success response for **play** request. A user *foo* playing with a user *bar*

message to user *foo*

SREQRP

```
{
        "type": "play",
        "result": "success",
        "opponent": "bar",
        "color": "white",
        "turn": 1,
        "msg": "Your turn. Start playing"
}
```

message to user *bar*

SREQRP

```
{
        "type": "play",
        "result": "success",
        "opponent": "foo",
        "color": "black",
        "turn": 0,
        "msg": "Not your turn. Wait your opponent's move"
}
```

Fail response to **play** request.

SREQRP

```
{
        "type": "play",
        "result": "fail",
        "msg": "you are put to waiting list",
}
```

Success response to **watch** request.

SREQRP

```
{
        "type": "watch",
        "result": "success",
        "boardstate": "in backgammon notation",
}
```

Fail response to **watch** request.

SREQRP

```
{
        "type": "watch",
        "result": "fail",
        "msg": "No match to watch",
        "msg": "I want to play",
        "msg": "I want to watch",
}
```

Success response for **leave** request.

SREQRP

```
{
        "type": "leave",
        "result": "success",
}
```

STDICE: Server Thrown Dice

STDICE is the server response to the client throw dice request (CTDICE).

The server is expected to reply with STDICE (Server Thrown Dice) when confronted with a client demanding the dice to be thrown.

Response Header: STDICE

Response Fields:

- dice: in backgammon notation

Example Response:

STDICE

```
{
	"dice": "in backgammon notation",
}
```

SRJCTM: Server Reject Move

SRJCTM is the server response to the active client's move request (CMOVEC).

The server is expected to reply with SRJCTM (Server Reject Move) when confronted with a passive client (who is waiting his/her opponent's move) demanding that his/her opponent made a wrong move.

Response Header: SRJCTM

Response Fields:

- boardstate: in backgammon notation

Example Response:

SRJCTM

```
{
	"boardstate": "in backgammon notation",
}
```

3.3. BC-BS Server Requests

SVPING: Server Ping

SVPING is the server heartbeat message sent to all the clients.

The client is expected to reply with CLPONG (Client Pong).

Request Header: SVPING

Expected Response: CLPONG

Example Request:

SVPONG

```
{
}
```

Example Response:

CLPONG

```
{
}
```

SMOVEC: Server Move Checker

SMOVEC is the active client's (whose turn to play) move sent by the server to all the interesting parties except the active client's himself/herself

The client is expected to reply with either CTDICE (Client Thrown Dice) or CRJCTM (Client Reject Move).

Request Header: SMOVEC

Request Fields:

- move: move in backgammon notation

Expected Responses: CTDICE or CRJCTM

Example Request:

SMOVEC

```
{
        "move": "in backgammon notation",
}
```

Example Responses:

CTDICE

```
{
}
```

CRJCTM

```
{
}
```

3.4. BC-BS Client Responses

CLPONG: Client pong

CLPONG is the client response to SVPING (Server Ping Request).

Response Header: CLPONG

Expected Response: CLPONG

Example Response:

CLPONG

```
{
}
```

CRJCTM: Client Reject Move

CRJCTM is the client response to SMOVEC (Server Move Checker). It is used to reject an opponent's (active player: whose turn to play) move. Backgammon clients issue the reject request after they see their opponent's move and decides that it is not a move which are allowed in backgammon rules.

Response Header: CRJCTM

Example Response:
CRJCTM

```
{
}
```

3.5. BC-BS Server Messages

These messages are sent by the server when particular conditions are held and they do not expect to bring any response from the clients.

SSTOVR: Server Set Over

SSTOVR is the server message to clients to inform that a set is over.

The clients are not expected to respond with any message.

Message Header: SSTOVR

Message Fields:
- winner: shows the username of the winner
- points: how many points are won by the user in this set

Example Message:

SSTOVR

```
{
        "winner": "foobar",
        "points": 1,
}
```

SGMOVR: Server Game Over

SGMOVR is the server message to clients to inform that game is over.

The clients are not expected to respond with any message.

Message Header: SGMOVR

Message Fields:
- winner: shows the username of the winner
- looser: shows the username of the looser

- winnerpoints: shows how many points winner got
- looserpoints: shows how many points looser got

Example Message:

SGMOVR

```
{
        "winner": "foobar",
        "looser": "nobody",
        "winnerpoints": 5,
        "looserpoints": 3,
}
```

STEARD: Server Teardown

STEARD is the server message to clients to inform that a player who is currently playing a match has lost its connection to the server (2 back-to-back SVPING messages) or leave the server.

The clients are expected to close their session.

Message Header: STEARD

Example Message:

STEARD

```
{
}
```

SVRNOK: Server Not OK

SVRNOK is the server message to clients to inform that they send a wrong request / response or at wrong time.

Message Header: SVRNOK

Example Message:

SVRNOK

```
{
}
```

References

1. SWECHAT Protocol: https://gist.github.com/canerturkmen/8115077

2. SWE544 lectures