Casino Blackjack Protocol (CBP) Design Paper

Stephen Hansen
June 4, 2021
CS 544

CONTENTS

1 Changes to Design

A number of changes have been made to the design of Casino Blackjack Protocol since the last submission, most of which were motivated by the implementation of the protocol. The changes are listed below.

The largest change to the protocol is that the split, surrender, and insurance PDUs have been removed. These features were not critical to the performance of CBP and the protocol could work fine without these PDUs to begin with. These were "nice to have" features, but development revealed that properly implementing these features correctly required much more programming work than anticipated. Even with these features removed, there are still well over 10 different PDUs between both the client and server, so the general project requirements should still be satisfied. The ability to enable surrender in the table settings has been removed altogether. I figure that these features could eventually make their way back into the protocol as part of "version 2" via the extensibility features mentioned in this document. The actions of "hit", "stand", and "double down" are still present in the design.

With these features removed, the DFA for the protocol becomes much simpler while still maintaining at least four states for the project requirements. The INSURANCE state and CHECK_BLACKJACK states are removed; the dealer no longer pre-checks for blackjack since there is no insurance bet. Likewise, START_TURN is removed and replaced with just TURN, as there are no actions like splitting which can only happen as the first action in a given turn. When it is time for a player's turn, the player transitions from WAIT_FOR_TURN to TURN. Once their turn is over (either by busting, standing, or a timeout), the state transitions to WAIT_FOR_DEALER.

The original design document mentioned a 10-second timeout if a player failed to act upon their turn in a given game. This timeout has been extended to 30-seconds so that the player has some time to think about their next move.

In the previous design, usernames and passwords had to be at least 9 characters long (including a LF at the end). This limitation on username/password length did not really make sense in the context of a protocol, so the lower limit on username/password length has been removed.

There was an issue with the previous design in that it was not clear whether the DFA was properly circular or not. I have added an arrow and a note to the DFA section explaining that on transitioning to the "DISCONNECTED" state (either by a literal disconnect or by the "QUIT" PDU), you technically transition back to the "VERSION" state (you start in this state when you re-connect). This should make the DFA circular and fix this issue.

Finally, a number of response codes were adjusted to make more sense in terms of implementation. The 3-1-2 response code which signifies start of turn has been changed from the "card hand response PDU" type to the "ASCII response PDU" type. A new response code, 2-1-5 has been introduced as a response for successfully leaving a table. This code falls under the ASCII response PDU and can be explicitly checked to make sure that the leave table request is handled. The 3-1-3 response code which denoted the dealer winning by blackjack has been removed, as this is effectively communicated in the 3-1-4 winnings PDU with a winnings amount of 0. Finally, the purpose of the 1-1-3 code has changed from "dealer reveals face card" (which happens automatically through a standard 1-1-1 response) to "dealt card to player/dealer, do not hit again". This is a response sent for a double down command, if the result of the action does not result in a 21 or bust. This helps the client distinguish between a response to a hit and a response to a double down.

2 Performance Implications

The protocol implementation works well with a large number of clients and messages. All clients run in separate threads so one client's behavior will not directly affect the other clients with regard to the server. However a large enough number of clients will increase the number of threads linearly, which could degrade overall performance between threads due to issues like context switching and increasing memory usage. In a worse case every client could be connected to a different game, so there would be n client threads and n game threads. Ideally the server should limit the number of games and number of overall clients at any given moment. Also, account and table information is stored by the server executable when this really should be part of some external database, because this could lead to inefficient memory usage and memory leaks with enough clients. But overall, in my testing, the performance works well; clients do not inhibit performance of other clients, and the server is concurrent in handling many messages at the same time.

3 Service Description

Casino Blackjack Protocol (CBP) is a network protocol designed to allow many users to play the popular casino game blackjack over the internet. CBP uses a model of having many clients connected to one server; the players of the game each use their own client to play, and the server acts as the casino managing multiple dealers. Beyond simply playing blackjack, CBP provides the ability to manage user funds and create "tables". A "table" is a specific instance of blackjack, with certain defined settings such as how the dealer plays, minimum and maximum betting limits, number of decks used, amount of players allowed at any given time, and at what rate you are paid if you win. A CBP server can host many different tables and a client may opt to join any table.

To play blackjack, a client connected and authenticated with a CBP server joins a specified room ID. If successful, the server notifies the client with details about the table's settings. If the player is the first person at the table, the game starts immediately. Otherwise, the incoming player must wait for the next round to start. Each player (client) starts by betting some amount. After some timeout, all players who bet are dealt two cards face-up, and the dealer (server) is dealt a face-up card and a face-down card. The goal of the game is for the player to have a higher hand value than the dealer, without going over 21 (called a bust). If the dealer busts, all players who did not bust win. In the case of a tie, the player gets their original bet back. Each card in the player's hand contributes some value. Turns go by player; on a player's turn, the player may choose to hit (request another card) or stand (stop requesting cards). Depending on the table, and the player's hand, the player may also be able to double down (double their bet, and only request one card). Once all players have made their turn, the dealer plays according to the table's policy as determined by the settings, and then awards players who win the game.

Part of what makes blackjack interesting when played live is that although the game takes place between a player and a dealer, other players at the table indirectly contribute to the player's game, despite no actual competition between players at the same table. Since a preset number of decks are shuffled together, the chance of a specific card appearing changes throughout the duration of the game. Seeing what cards the other players have drawn can affect the likelihood of your next card and the dealer's next card, which presents some interesting strategies. Card counting, where a player tracks the cards that have been drawn, gives the player advantage over the dealer. Strategies like card counting are often banned by casinos, since it causes the casino to lose money! CBP provides no mechanism for detecting card counting, but to make the protocol as accurate to live blackjack as possible, CBP provides messages to broadcast actions done by each player to all of the other players at the table. CBP also includes very basic chat functionality for players at the same table to talk with each other.

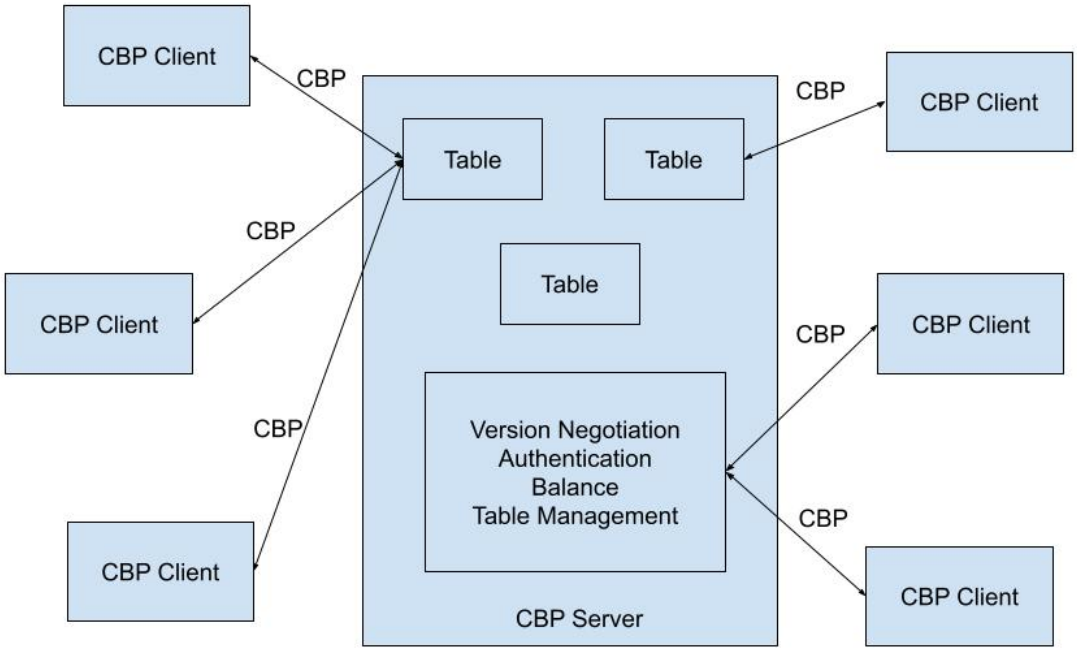
CBP is meant to work over TCP/IP as an application layer protocol, with TCP as a reliable transport layer. A CBP server uses a well-known port number of 21210. A client first connects to a CBP server and does a version negotiation to ensure that the client's version of CBP is supported by the server. Then, the client authenticates through a username command and password command. CBP should be used over TLS (transport layer security), as CBP provides no encryption for security. Once authenticated, users may add funds to their account and play in blackjack tables. Users may also have the option to add or remove tables, with ability to customize details of each table.

CBP is designed to be very extensible. Blackjack is a very popular casino game, and casinos are always trying to create new rules and variations of blackjack to attract players. The design of the client and server PDU headers is such that CBP may, in the future, support casino games other than blackjack (thus becoming an overall Casino Game Protocol). Table customization is highly extensible, and CBP is specifically designed so that the server handles all game logic, such that clients may be able to participate in games with new settings without requiring any explicit update.

The functionality provided by CBP can be described as follows:

- Version negotiation
- Authentication
- Casino balance management
- Table management
- Game logic (blackjack)
- Chat/event broadcasting

CBP Client/Server Diagram



4 Message Definition (PDUs)

The PDUs for CBP are broken into two categories: client commands and server responses. First, I'll denote here what the command field types are, and how they are represented in memory. Then, I'll detail the PDU for each client command, and finally detail the PDU for each server response.

4.1 Data Types

The following data types are defined and used in CBP. For reference throughout this document, a byte is defined as being 8 bits in length.

4.1.1 Unsigned int

This is a standard, unsigned integer, where each bit represents the base 2 encoding of the integer. In case the unsigned int takes up multiple bytes, the bytes are ordered big endian (commonly referred to as network order), where the most significant byte appears first in the message and the least significant byte appears last.

4.1.2 Signed int

Like the unsigned integer, the signed integer will be ordered big endian if the integer takes up multiple bytes. The signed integer is represented in two's complement, where the most significant bit of the most significant byte represents the sign (0 for positive, 1 for negative), and the rest of the bits are the base 2 encoding assuming the integer is positive. To negate a positive integer in two's complement, you take all the bits, invert them, and add 1.

4.1.3 8-bit ASCII character

This is an 8-bit field which, depending on the bits, corresponds to a certain character per ASCII character encoding. Since ASCII encoding only uses 7 bits, the most significant bit is always going to be 0, and the remaining bits will be the ASCII character encoding. One byte corresponds to one 8-bit ASCII character.

4.1.4 8-bit ASCII character string

An 8-bit ASCII character string is a sequence of 8-bit ASCII characters contiguous to each other in memory (an array of characters, in other words). The characters are ordered in the direction that they should be parsed; for example, the string "hello world" will begin with the 8-bit ASCII character (byte) for "h", followed by the ASCII character (byte) for "e", ..., up to the ASCII character (byte) for "d". Termination of the string will vary by the PDU; all PDUs place a limit on the length of the string, and most terminate the string with the ASCII linefeed (LF) character.

4.1.5 CARD ENUMERATION

A card enumeration consists of two 8-bit ASCII characters and is therefore two bytes in length. The first character denotes the rank. The characters and ranks are defined as follows:

Ranks and values

Character	Rank	Soft value	Hard value
A	Ace	11	1
2	Two	2	2
3	Three	3	3
4	Four	4	4
5	Five	5	5
6	Six	6	6
7	Seven	7	7
8	Eight	8	8
9	Nine	9	9
T	Ten	10	10
J	Jack	10	10
Q	Queen	10	10
K	King	10	10

The value of a hand is computed by summing the values of each rank. The algorithm for computing the hand value is as follows. Sum the soft values of each card in the hand. If the sum of soft values is 21 or less, the sum is the hand's value. Otherwise, sum the hard values of each card in the hand. The sum of hard values is the hand's value.

The second character denotes the suit. The suits are defined as follows:

Suits

Character	Suit
D	Diamonds
C	Clubs
H	Hearts
S	Spades

The suit has no contribution on the hand value. These two characters fully define a card; for example "7H" represents the seven of hearts.

4.2 CLIENT COMMANDS

CBP client command PDUs are specific to the type of command. Each command begins with an 8-bit unsigned integer code indicating the category. This integer code is followed by another 8-bit unsigned integer code representing the code. The reasoning for this distinction between category and command will be explained later in Section 4. This second code is then followed by a variable length of data which depends on the command code. Here is the list of all accepted commands for version 1 of CBP.

All supported commands in CBP Version 1

Category code	Command code	Command name
0	0	VERSION
0	1	USER
0	2	PASS
0	3	GETBALANCE
0	4	UPDATEBALANCE
0	5	QUIT
1	0	GETTABLES
1	1	ADDTABLE
1	2	REMOVETABLE
1	3	JOINTABLE
1	4	LEAVETABLE
1	5	BET
1	7	HIT
1	8	STAND
1	9	DOUBLEDOWN
1	12	CHAT

Below, I go through each supported command and what the PDU structure looks like for each.

VERSION command

Field Name	Offset (bits)	Length (bits)	Type	Description
Category code	0	8	Unsigned int	Category code 0 for general usage.
Command code	8	8	Unsigned int	VERSION has command code 0.
Version	16	32	Unsigned int	Version number ID.

The VERSION command MUST be sent by a client as the first message when the client connects to a CBP server. In the message, the client includes the current version of the protocol that it is running so that the server can determine whether it can support the client or not. Due to the extensibility of the protocol, it is possible that CBP might have many versions that add new features or versions specific to certain casinos (if a casino wanted to add specific functionality or restrict users to use the casino's defined client application). Because of this, the version number field is intentionally four bytes long, so that it can support a large enough range (over 2 trillion) of possible versions.

Version 1 of CBP (as defined by this document) requires sending the VERSION command with an unsigned integer value of 1.

USER command

Field Name	Offset (bits)	Length (bits)	Type	Description
Category code	0	8	Unsigned int	Category code 0 for general usage.
Command code	8	8	Unsigned int	USER has command code 1.
Username	16	Between 8 and 264	8-bit ASCII character string	Username for authentication.

Once the server responds with a response indicating that the client's protocol version is supported, the client MUST next send the USER command. This command provides the server with the username of the user running the client. The length of the username string must be between 72 (9 characters) and 264 (33) characters long. The length will be some multiple of bytes, or 8 bits. The username is terminated with an ASCII LF character; this character counts towards the length of the ASCII character string, but is NOT to be included in the parsed username (so

really the username is between 8 to 32 characters long). This means that LF cannot be used in the middle of an actual username. If 33 or more characters are read by the server, and no LF is encountered, the message sent is NOT a valid USER command and WILL BE discarded by the server. The server will respond with a successful reply if the username meets the length requirements. Any additional requirements (such as not allowing certain ASCII characters in the username) can be determined on a per-server basis, but this protocol makes no requirement on character restrictions beyond that LF may not be part of any username.

This standard of limiting a string field to a certain limit, not allowing LF as part of the string, and requiring the string to terminate with LF (including the LF character in the length limit) will appear through multiple messages in this protocol. To avoid repetition, I refer to this standard as a string being "LF-termination compliant".

PASS command

Field Name	Offset (bits)	Length (bits)	Type	Description
Category code	0	8	Unsigned int	Category code 0 for general usage.
Command code	8	8	Unsigned int	PASS has command code 2.
Password	16	Between 8 and 264	8-bit ASCII character string	Password for authentication.

Once the username command is sent and the client receives a successful reply, the client MUST send the PASS command next. This command is identical to the USERNAME command except this time the password is sent. Like before, this password is LF-termination compliant and is between 9 to 33 characters long (including LF). The password is NOT encrypted by the protocol; CBP does no encryption itself, but TLS MUST be used to encrypt all messages. This ensures security so that the password cannot be retrieved by packet-sniffing the network. The password sent should be the corresponding password for the username sent previously. If successful, the client will receive a successful reply and is officially "authenticated"; the client may now begin adjusting funds or playing games. A negative reply indicates restarting the entire authentication sequence beginning with the USERNAME command; this reply does NOT indicate whether the username or password failed, to discourage oracle attacks.

GETBALANCE command

Field Name	Offset (bits)	Length (bits)	Type	Description
Category code	0	8	Unsigned int	Category code 0 for general usage.
Command code	8	8	Unsigned int	GETBALANCE has command code 3.

The GETBALANCE command may be sent by an authenticated client to retrieve the funds stored by the CBP server for the current user. The reply, if successful, will include the amount of funds in the reply PDU. This command should be usable in the middle of any game so that the user can see what the current balance is (in case the user is running out of money).

UPDATEBALANCE command

Field Name	Offset (bits)	Length (bits)	Type	Description
Category code	0	8	Unsigned int	Category code 0 for general usage.
Command code	8	8	Unsigned int	UPDATEBALANCE has command code 4.
Funds	16	32	Signed int	Amount of funds to add or remove.

The UPDATEBALANCE command may be sent by an authenticated client to add or remove funds to their current balance. The funds are added or removed relative to the current balance; sending UPDATEBALANCE with a funds amount of 1 will add (deposit) 1 to whatever the current balance is (and NOT necessarily set the balance to 1). Similarly, if the funds amount is -1, this will remove (withdraw) 1 from the current balance. The response code will indicate whether the update was successful or not. This update can fail if, say, adding the funds would cause an overflow in the balance, or if removing the funds would result in a negative balance. Like GETBALANCE, this command should be usable in the middle of any game.

QUIT command

Field Name	Offset (bits)	Length (bits)	Type	Description
Category code	0	8	Unsigned int	Category code 0 for general usage.
Command code	8	8	Unsigned int	QUIT has command code 5.

The QUIT command terminates the connection between the client and server. This command may be used at any point, does not require authentication, and may even be used in the middle of a game. For blackjack, this requires notifying the other players (if any) that the user has left, and skipping (or ending) the user's turn, based on the current situation of the game.

GETTABLES command

Field Name	Offset (bits)	Length (bits)	Type	Description
Category code	0	8	Unsigned int	Category code 1 for blackjack.
Command code	8	8	Unsigned int	GETTABLES has command code 0.

GETTABLES is the first blackjack-specific command here as denoted by it having a category code of 1. This command requests the server to send the client a list of blackjack tables. If successful, the client will receive back a list of unique table IDs. This list will also denote settings specific to each table, so that the user may join a table that is preferred. The list of table settings, and format of these settings, will be detailed later.

ADDTABLE command

Field Name	Offset (bits)	Length (bits)	Type	Description
Category code	0	8	Unsigned int	Category code 1 for blackjack.
Command code	8	8	Unsigned int	ADDTABLE has command code 1.
Settings	16	Between 16 and 8208	8-bit ASCII character string	A list of zero or more headers.

ADDTABLE provides a way for tables with certain settings to be created. Depending on server implementation, this command could be usable by users designed as admins, or by any client connected to the server. CBP places no hard requirement on determining which users have access to ADDTABLE and REMOVETABLE; this is up to the designer of the server. The ADDTABLE command supports a number of settings, and is extensible for additional settings. The settings field may be between 16 (two ASCII characters) and 8208 (1026 ASCII characters) bits long. The settings field MUST be terminated with two back-to-back line feeds (LF, character code 10) (this allows 0-1024 characters for actual data). The limitations from before for LF-termination compliant strings apply here; a string equal to or longer than 1026 characters (not including LFLF) is not allowed for this message, and the message is effectively ignored by the server.

The contents of settings are formatted as follows (in BNF notation). Treat <LF> as the ASCII linefeed character.

```
<settings>      ::= <header-list> | <LF><LF>
<header-list>   ::= <header-value><header-list> | <header-value><LF>
<header-value>  ::= <header>:<value><LF>
```

I define `<header>` as an ASCII string of 1 or more characters, any character is allowed except : (colon) (character code 58) and LF. `<value>` can be defined similarly, an ASCII string of 1 or more characters, any character allowed except : (colon) and LF. This format for the settings is similar to that used for the headers in HTTP, and I will refer back to this format frequently in other PDUs. It is unambiguous; the setting list terminates in two subsequent line feeds. Single line feeds are used to delimit between header/value pairs, and colons are used to delimit between headers and their corresponding values. All headers, values are ASCII strings - meaning you cannot intermix an unsigned integer in the middle of the settings list - all number values must be represented as ASCII (so the number 312 would take up three characters, or bytes, for "3", "1", and "2"). This is for ease of parsing, and extensibility; the entire contents can be interpreted as ASCII, and then any headers can be converted to a more suitable type if necessary.

Now, I define the standard headers that are supported in the first version of CBP.

Header Name	Value Format	Description	Default
max-players	<max-number-of-players>	Set how many players can join the table.	max-players:5
number-decks	<number-of-decks>	Number of decks shuffled.	number-decks:8
payoff	<pay-amt>-<recv-amt>	Payoff ratio for winners.	payoff:3-2
bet-limits	<min-bet>-<max-bet>	Minimum and maximum bet amounts.	bet-limits:25-1000
hit-soft-17	<true-or-false>	True if dealer hits on soft 17. If false dealer stands.	hit-soft-17:true

Each header need only be provided once or not at all. The example values provided by the table above should be used as default settings for each header that is not provided. If any header is sent with an improperly formatted value or invalid value (e.g. sending `bet-limits:1000-25`), the server will respond with an appropriate error (for example 5-1-0 for blackjack command failure). Fields like `<max-number-of-players>`,`<number-of-decks>`,`<pay-amt>`, `<recv-amt>`,`<min-bet>`,`<max-bet>` may only consist of a sequence of integers in ASCII (`0,1,2,3,4,5,6,7,8,9`, no other characters allowed). Unrecognized headers will be ignored by the server but will not directly result in an error response, unless any recognized headers have invalid values. If a header is sent more than once, the server will respond with a 5-1-0 blackjack command failure response (ambiguous as to which header value to use). The generic format of the settings is very extensible to the addition of new features, and this will be discussed later in Section 4.

On successful creation of a table, the user will be sent back a successful response code, with the unique ID of the table included in the response.

REMOVETABLE command

Field Name	Offset (bits)	Length (bits)	Type	Description
Category code	0	8	Unsigned int	Category code 1 for blackjack.
Command code	8	8	Unsigned int	REMOVETABLE has command code 2.
Table ID	16	16	Unsigned int	The unique ID of the table to remove.

REMOVETABLE removes the table with the given ID. Each table, on creation, is assigned a unique table ID to distinguish tables from each other. This is the same ID that is provided by the response to LISTTABLES and the same ID provided on a successful response to ADDTABLE. REMOVETABLE should only be successful for the same users who have access to ADDTABLE. If the ID provided is valid, the table will be deleted from the listing, and all players currently at the table (if any) will receive a server response indicating that they are being kicked out of the table. On an invalid ID, the user will get back an error response.

JOINTABLE command

Field Name	Offset (bits)	Length (bits)	Type	Description
Category code	0	8	Unsigned int	Category code 1 for blackjack.
Command code	8	8	Unsigned int	JOINTABLE has command code 3.
Table ID	16	16	Unsigned int	The unique ID of the table to join.

JOINTABLE may be used by authenticated users to join a game of blackjack. The table ID provided is the unique ID of the table to join. Once at the table, the player will have access to some more blackjack commands once a new round starts. If a client sends a JOINTABLE request while at an existing table, the client will receive an error response. The LEAVETABLE command must be used prior to joining a new table.

LEAVETABLE command

Field Name	Offset (bits)	Length (bits)	Type	Description
Category code	0	8	Unsigned int	Category code 1 for blackjack.
Command code	8	8	Unsigned int	LEAVETABLE has command code 4.

A authenticated user at a table may send a LEAVETABLE command to leave the current table. This places the user back into the general authentication state where they may list tables and are not currently playing at any individual table.

BET command

Field Name	Offset (bits)	Length (bits)	Type	Description
Category code	0	8	Unsigned int	Category code 1 for blackjack.
Command code	8	8	Unsigned int	BET has command code 5.
Bet amount	16	32	Unsigned int	Amount of funds to bet.

Once at a table and a round has started, the player may place a bet to participate. The bet amount is sent as an unsigned integer of 4 bytes. The user may only bet within the allowed bet range at the table and may not bet an amount that is larger than the current balance. A successful reply to this command indicates that the user is playing in the current round and the bet has been deducted from the balance. A negative reply indicates that the bet was unsuccessful, and not deducted from the player's balance. You may only have one bet with a successful reply per each round.

HIT command

Field Name	Offset (bits)	Length (bits)	Type	Description
Category code	0	8	Unsigned int	Category code 1 for blackjack.
Command code	8	8	Unsigned int	HIT has command code 7.

On a player's turn, the player may send a HIT command to request another card. A successful response will include the new card and the player's new total hand value.

STAND command

Field Name	Offset (bits)	Length (bits)	Type	Description
Category code	0	8	Unsigned int	Category code 1 for blackjack.
Command code	8	8	Unsigned int	STAND has command code 8.

On a player's turn, the player may send a STAND command to stop requesting cards. A successful response indicates end of turn and the server moves to the next player's turn, or to the dealer's turn if all participating players (players with bids) have finished.

DOUBLEDOWN command

Field Name	Offset (bits)	Length (bits)	Type	Description
Category code	0	8	Unsigned int	Category code 1 for blackjack.
Command code	8	8	Unsigned int	DOUBLEDOWN has command code 9.

On a player's turn, the player may send a DOUBLEDOWN command. This command results in a few events. First, the player's original bid is doubled. If the player cannot afford to double the bid, the command fails and a negative response is returned. The player is then dealt a new card, and receives the card (along with the new hand total) through another response. Finally, the player is forced to stand - effectively, the player's turn ends.

CHAT command

Field Name	Offset (bits)	Length (bits)	Type	Description
Category code	0	8	Unsigned int	Category code 1 for blackjack.
Command code	8	8	Unsigned int	CHAT has command code 12.
Text	16	Between 16 and 1032	8-bit ASCII character string	Text to send to other players.

The CHAT command is fairly simple. At any point during the game at a table, you may send a message to the other players through the CHAT command. The length of the text sent is between 16 (two ASCII characters) and 1032 (129 ASCII characters) bits long. This text is LF-termination compliant.

4.3 Server Responses

Server responses begin with a generic header. This header is made up of three bytes, each byte representing an unsigned integer. The first byte indicates the type of response; whether it is positive, negative, and whether it is permanent, intermediate, transient or preliminary. The designations of this first byte are borrowed from FTP which did a reasonable job of assigning reply codes to specific categories. The second byte corresponds to the category code; 0 for general responses and 1 for blackjack-specific responses. The third byte denotes a specific message. After the three byte header, the rest of the PDU data is determined by the reply code.

First Reply Code Categories

First Reply Code	Meaning
1	Positive Preliminary (expecting another command or response)
2	Positive Completion (action was successfully completed)
3	Positive Intermediate (command accepted, need another command)
4	Transient Negative Completion (action failed, try again)
5	Permanent Negative Completion (action failed, do NOT try again)

Here is the list of reply codes, with their general meanings.

Reply Code Meanings

First Reply Code	Second Reply Code	Third Reply Code	Meaning
3	0	0	Provide password
2	0	0	General command success
4	0	0	General command tentative failure
5	0	0	General command failure
5	0	1	Unsupported version
5	0	2	Unknown command
2	0	1	Supported version
2	0	2	Authenticated successfully
5	0	2	Authentication failure
2	0	3	Balance provided
2	1	0	Blackjack command success
4	1	0	Blackjack command tentative failure
5	1	0	Blackjack command failure
5	1	1	Command not allowed by table
2	1	1	Sent list of tables
4	1	1	No tables available
4	1	2	Table with ID does not exist
4	1	3	Table provided is full, try again later
3	1	0	Table status sent, provide bet(s)
1	1	0	Game-in-progress, please wait for next round
4	1	4	Table is being closed
1	1	1	Dealt card to player/dealer
1	1	2	Dealt card to player/dealer, bust
1	1	3	Dealt card to player/dealer, do not hit again
1	1	4	Player/dealer obtained blackjack
3	1	2	Player's turn
1	1	5	Broadcast player/dealer event
3	1	4	Player wins amount, provide bets
2	1	4	Table added, ID provided
1	1	6	Player obtains 21 (not blackjack)
1	1	7	Timeout elapsed
2	1	5	Left table

Now, I will go through the exact reply code PDU structures of data, and group reply codes by each structure. I use the notation of "f-s-t" to denote a specific reply code, where "f" is the first reply code value, "s" is the second reply code value, and "t" is the third reply code value. First, the most common PDU for a response:

ASCII Response Data PDU

Field Name	Offset (bits)	Length (bits)	Type	Description
First reply code	0	8	Unsigned int	Overall response type.
Second reply code	8	8	Unsigned int	Corresponds to a category code.
Third reply code	16	8	Unsigned int	Corresponds to an exact response type.
Data	24	16 or more	8-bit ASCII string	Extra information sent for the message. Ends in LFLF.

ASCII Response Data PDU responses: 3-0-0, 2-0-0, 4-0-0, 5-0-0, 5-0-2, 2-0-2, 5-0-2, 2-1-0, 4-1-0, 5-1-0, 5-1-1, 4-1-1, 4-1-2, 4-1-3, 1-1-0, 3-1-2, 4-1-4, 1-1-5, 1-1-7, 2-1-5

These responses indicate meaning through the reply code, and technically do not require sending any additional information in the data field. An empty data field is signified by data just containing LFLF (LFLF immediately folowing the three byte code header). The data field may be used instead to communicate a specific reason for the response code. For example, response 5-0-2 might have a data field with ASCII contents "Error, unknown command sent by client<LFLF>". CBP does not define specifically what the data field should be for each of these messages (this is up to server implementation) but servers can provide user-friendly information in this field for these responses, such that the client can parse it out and display to the user interface. Different data field contents may be used for the same response code. The message broadcast reply code (1-1-5) can be used to tell other players at the table about the current player's turn. For example, a 1-1-5 can be sent with data "Player 3 stands" or "Player 3 hits KH, busts". Chat messages must be sent through this 1-1-5 response but exact formatting (and frequency of broadcast responses) is up to server implementation. The DFA in section 3 shows specifically how the protocol enforces that players make progress in blackjack and how they are notified of their own cards and the dealer's, but no hard requirement is placed on communicating the player's moves to other clients at the same table. The option to do so is provided by the protocol, but is not mandatory for CBP to function. The only hard requirement here is that the data itself may not contain LFLF anywhere, otherwise it becomes ambiguous for the client to determine where the data ends.

Version Response Data PDU

Field Name	Offset (bits)	Length (bits)	Type	Description
First reply code	0	8	Unsigned int	Overall response type.
Second reply code	8	8	Unsigned int	Corresponds to a category code.
Third reply code	16	8	Unsigned int	Corresponds to an exact response type.
Version	24	32	Unsigned int	Version ID of the server.

Version Response Data PDU responses: 5-0-1, 2-0-1

The client sending the VERSION command receives a response on version negotiation (see the DFA); 5-0-1 is sent by the server if the client's version is not supported by the server, and 2-0-1 is sent by the server if the client's version is supported. The data after the reply code denotes the server's version. The size and type of the version field is the same as the client's. A client receiving the version response PDU with an equal version from the server indicates that both the client and server are running the same version of CBP.

Balance Response Data PDU

Field Name	Offset (bits)	Length (bits)	Type	Description
First reply code	0	8	Unsigned int	Overall response type.
Second reply code	8	8	Unsigned int	Corresponds to a category code.
Third reply code	16	8	Unsigned int	Corresponds to an exact response type.
Balance	24	32	Unsigned int	The current authenticated user's balance.

Balance Response Data PDU response: 2-0-3

The GETBALANCE command sent by an authenticated user must return the 2-0-3 response from the server. The data following the header reply code is the user's balance, in 32 bits, as an unsigned integer.

LISTTABLES ASCII Response Data PDU

Field Name	Offset (bits)	Length (bits)	Type	Description
First reply code	0	8	Unsigned int	Overall response type.
Second reply code	8	8	Unsigned int	Corresponds to a category code.
Third reply code	16	8	Unsigned int	Corresponds to an exact response type.
Number of tables	24	16	Unsigned int	Number of tables hosted by server.
Variable data	40	0 or more	Multiple Tabledata	See explanation.

Tabledata PDU

Field Name	Offset (bits)	Length (bits)	Type	Description
Table ID	0	16	Unsigned int	Unique ID of table instance
Settings Data	32	16 or more	8-bit ASCII string	Settings for given table ID.

LISTTABLES ASCII Response Data PDU response: 2-1-1

This is the most complicated PDU, so I have to explain this carefully. The LISTTABLES response, if successful, will return response code 2-1-1 to a user and the response provides a list of tables with the settings for each table. The 2-1-1 response PDU is structured as follows. Following the three byte reply code, the next two bytes denote the number of tables available. Call the number of tables available n, The variable data following these two bytes will be n tabledata PDUs. In other words, the number of tables tells the client to parse out that many tabledata PDUs; this is a sub-PDU contained inside the response PDU. The tabledata PDU is structured as follows. The table ID makes up the first two bytes. The settings for that table ID immediately follow. The settings are an 8-bit ASCII string structured as per the BNF grammar described for the headers in ADDTABLE. In other words the settings data is the headers that were provided when creating the table with ADDTABLE. The end of the tabledata PDU is denoted by LFLF. Thus to parse out the entire LISTTABLES response PDU, given n tables, the tabledata PDU structure must parse out the table ID and ending LFLF n times. The grammar is provided again below for reference. Like in ADDTABLE, if a header is missing from the settings data, then the table has the default value for that header as defined by this document.

```
<settings>      ::= <header-list> | <LF><LF>
<header-list>   ::= <header-value><header-list> | <header-value><LF>
<header-value> ::= <header>:<value><LF>
```

For example, given a server that hosts two tables, one with ID 1 and another with ID 1234, the first tabledata PDU can hold ID 1 and the second can hold 1234. The first tabledata might have settings data formatted as follows:

```
max-players:7<LF>
payoff:3-2<LF>
bet-limits:5-50<LF>
<LF>
```

Assume defaults for the missing settings. So the number of decks is 8 and the dealer hits on soft 17. The second tabledata may be formatted as follows:

```
payoff:6-5<LF>
hit-soft-17:false<LF>
<LF>
```

For the missing settings, use the defaults: maximum number of players is 5, number of decks shuffled is 8, and bet limits are 25 to 1000. The tabledata for table 1 is listed first with the settings shown here, then the tabledata for table 1234 with the settings shown here.

CBP does not require any specific ordering of the tabledata in the response. Just that each created table is provided once in the response with unique IDs per each table.

ADDTABLE Response Data PDU

Field Name	Offset (bits)	Length (bits)	Type	Description
First reply code	0	8	Unsigned int	Overall response type.
Second reply code	8	8	Unsigned int	Corresponds to a category code.
Third reply code	16	8	Unsigned int	Corresponds to an exact response type.
Table ID	24	16	Unsigned int	ID of newly created table.

ADDTABLE response: 2-1-4

On a succesful ADDTABLE command, the server responds with 2-1-4. This message includes the ID of the newly added table in the response. This table ID along with the settings sent in the initial command can now be provided as a new tabledata PDU in the LISTTABLES response.

JOINTABLE Response Data PDU

Field Name	Offset (bits)	Length (bits)	Type	Description
First reply code	0	8	Unsigned int	Overall response type.
Second reply code	8	8	Unsigned int	Corresponds to a category code.
Third reply code	16	8	Unsigned int	Corresponds to an exact response type.
Settings Data	24	16 or more	8-bit ASCII string	Settings for joined table.

JOINTABLE response (round started): 3-1-0

On joining a table, for the first round joined, the client is sent the table's specific settings per the 3-1-0 response. The "settings data" is defined using the same BNF grammar for the ADDTABLE command and LISTTABLES 2-1-1 response. Since the settings data terminates in LFLF, the PDU is not different from the ASCII Response Data PDU in terms of structure, but has this grammar baked into the data.

Card Hand Response Data PDU

Field Name	Offset (bits)	Length (bits)	Type	Description
First reply code	0	8	Unsigned int	Overall response type.
Second reply code	8	8	Unsigned int	Corresponds to a category code.
Third reply code	16	8	Unsigned int	Corresponds to an exact response type.
Holder	24	8	Unsigned int	Denotes whether hand is player's or dealer's
Soft Value	32	8	Unsigned int	Value of the hand, by soft valuation
Hard Value	40	8	Unsigned int	Value of the hand, by hard valuation
Number of cards	48	8	Unsigned int	Number of cards in the hand
Cards	56	0 or more	Card enumeration	The exact cards

Card hand responses: 1-1-1, 1-1-2, 1-1-3, 1-1-4, 1-1-6

These responses indicate a game condition and send back either the entire player's (client's) hand or the dealer's. Following the first three bytes, the fourth byte denotes whether the hand belongs to the player or dealer. A value of 0 here means the dealer, and any non-zero value means the player. The fifth byte is the soft value of the hand. This is computed by adding the soft value of each card in the hand (defined in the card enumeration type). The hard value (sixth byte), likewise, is computed by adding the hard value of each card in the hand. The number of cards is the seventh byte, which specifies the size of the hand, and is necessary to parse the end of the PDU. The rest of the PDU data is a series of card enumerations. Card enumerations are defined under the data types listed previously; each enumeration is two ASCII characters (so two bytes per card). Given n cards, the data following the number of cards byte is $2 * n$ bytes representing each individual card. For example, if $n = 3$, and the cards after are in ASCII "AH2D3C", this represents an ace of hearts ("AH"), two of diamonds ("2D"), and three of clubs ("3C"). Every response code that is a card hand PDU MUST send the ENTIRE hand as relevant to either the player or dealer per the PDU. If the player hits, or starts the turn with a 3-1-2 response, the PDU sent back MUST be with regard to the player (NOT the dealer). The entire hand must be sent; a HIT may not result in a card hand response that only contains the newly hit card; the entire hand must be sent so that the client knows the new soft and hard values of their hand. The dealer's face-down card does NOT count in the dealer's valuations and may NOT be included in the number of cards sent, until it is turned face-up.

Winnings Response Data PDU

Field Name	Offset (bits)	Length (bits)	Type	Description
First reply code	0	8	Unsigned int	Overall response type.
Second reply code	8	8	Unsigned int	Corresponds to a category code.
Third reply code	16	8	Unsigned int	Corresponds to an exact response type.
Winnings	24	32	Unsigned int	Amount of funds won in current round

Winnings responses: 3-1-4

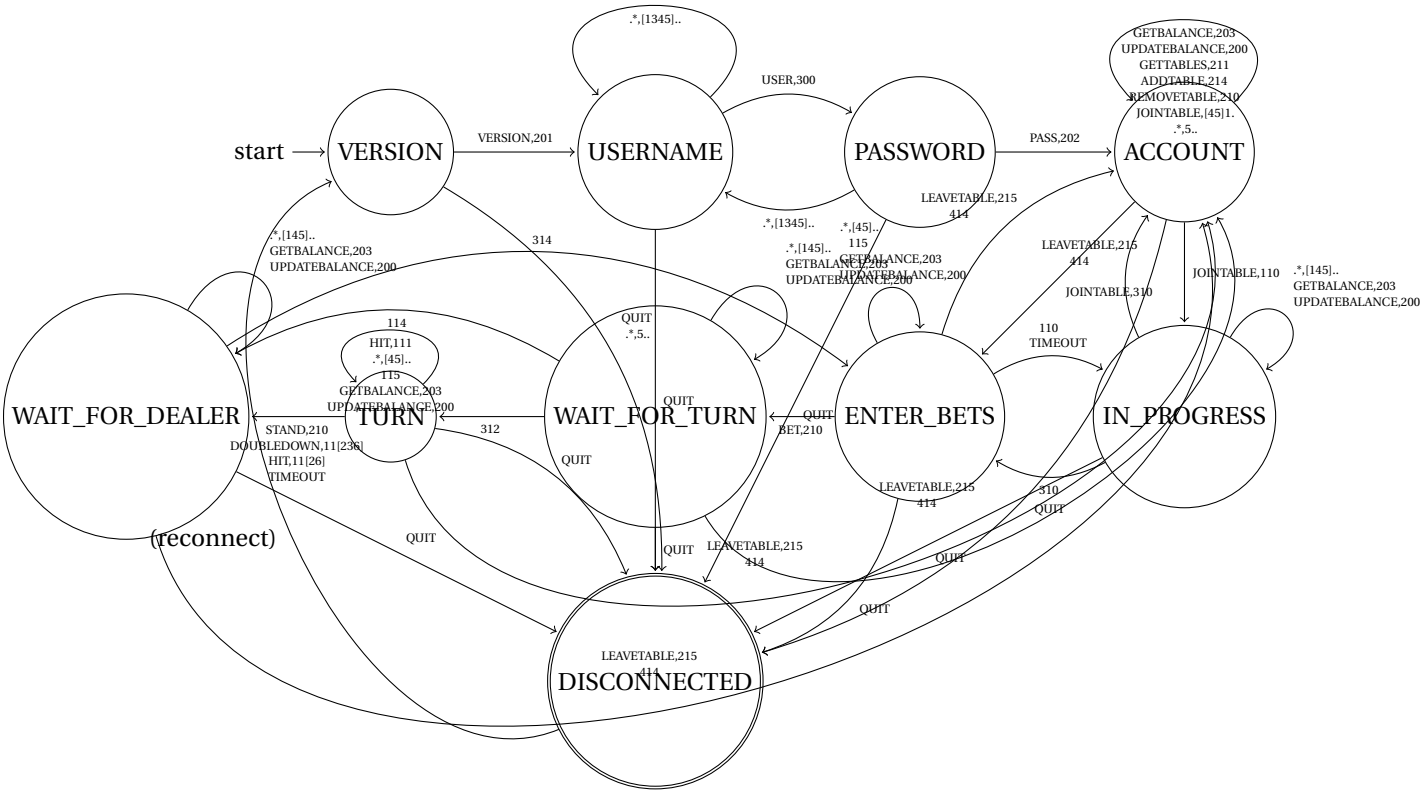
The final PDU described here is for messages that end the round and communicate to the players what they win. The initial loss (the loss of the bet) is not part of the winnings calculation here. If a player wins nothing (they lose their bet, in other words), the value of winnings is 0. In a table with 3-2 payoff, if a player wins against the dealer, the player will get 1.5 times their original bet, and that value will be sent in winnings. If the winnings results in a floating point number, just round the number up to the nearest whole integer (so that the winnings can be sent as an unsigned int). The winnings are also immediately added to the player's balance on the server when this response is sent.

5 DFA

Provided is a DFA of CBP which is accurate to how the protocol operates. However, there is a large amount of edges, which is explained below the DFA. I have supplied a second version of this DFA after this one, with a number of the edges removed, to make the DFA more readable.

For notation, I list the commands and responses at each edge. Commands and responses that share the same edge are listed on multiple line. Regular expression notation is used to match against multiple response codes or commands. A comma indicates a client command and expected server response. Conveniently, since none of the reply codes have a number with two or more digits (in base 10), I list the three codes next to each other in the diagram, to save space. For example "300" refers to "provide password" and is analogous to writing about response code "3-0-0". Also, the word "TIMEOUT" appears a couple times in the DFA. This is not a command; this indicates in all cases, a 30-second timeout on the server reading a message from the client. The timeout elapsing is indicated to the client through the "1-1-7" response code. This is to prevent a player from stalling a table from making progress, as otherwise if any one player did nothing during their turn, the server would wait indefinitely for the player's action. This will be referenced again in Section 5.

CBP DFA with all edges

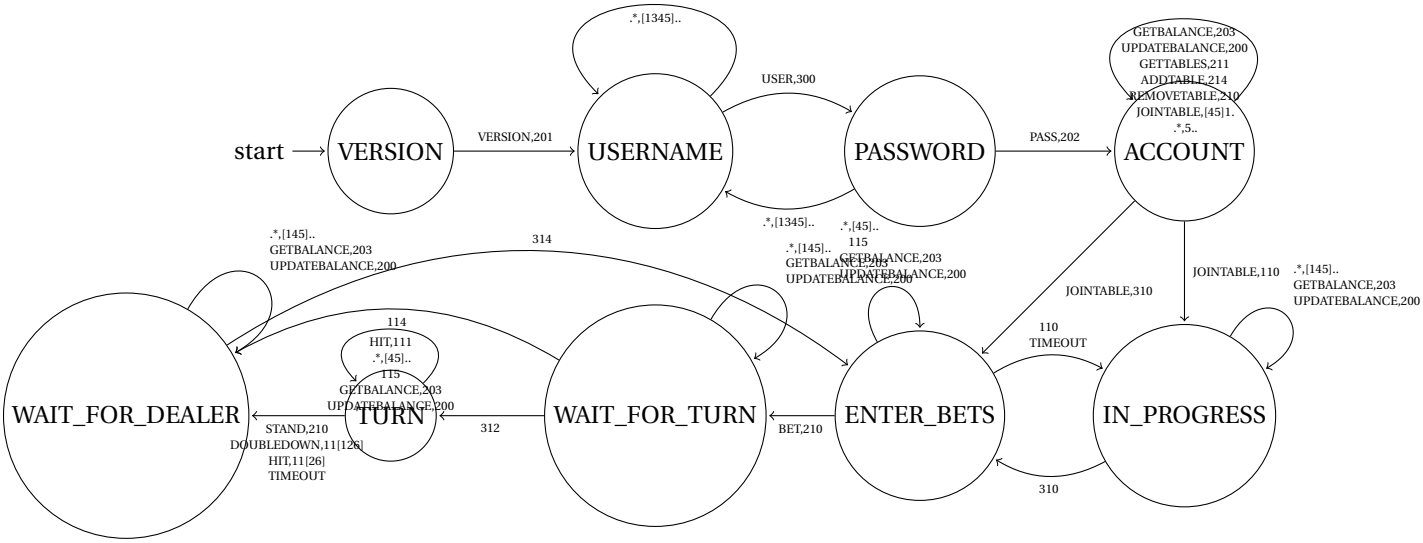


There is an arrow from the DISCONNECTED state to the VERSION state. This arrow signifies that on disconnect, when the client re-connects to the server, the protocol restarts back at the VERSION state, thus making this DFA circular.

This DFA is messy for four reasons. First, once a user (client) joins a game via JOINTABLE, the user may leave at any point by issuing the LEAVETABLE command, and receiving a 2-1-5 response. The states in which the user is "at" the table are IN_PROGRESS, ENTER_BETS, WAIT_FOR_TURN, INSURANCE, START_TURN, TURN, and

WAIT_FOR_DEALER. Second, the user might be kicked out of a table if a different user with sufficient privileges issues the DELETETABLE command with the ID of the table that the user is at. In this case, the user receives a response code of 4-1-4 and knows now that the protocol is back at the ACCOUNT state, because the table has been closed. This is indicated through the same edge for the LEAVETABLE command. Third, at ANY state (except DISCONNECTED), the user may issue the QUIT command to disconnect. This command does not require waiting on any specific server response. This immediately moves the protocol DFA to the DISCONNECTED state. Finally, at the VERSION state, if the user sends the VERSION command and receives a 5-0-1 response, it means that the version is not supported and the client is disconnected from the server. Any command other than VERSION sent here will also receive a 5-0-0 response and the client will be disconnected. Because of these three situations, there are many edges on this DFA which communicate these situations but make the DFA highly unreadable. The next DFA diagram is the same as this one, but the DISCONNECTED state is removed, and the edges for LEAVETABLE, 4-1-4, QUIT, and 5-0-1 are all removed to improve readability.

CBP DFA with disconnect and leave table edges removed



The DFA for CBP operates as follows. All clients on connecting to the server first start in the VERSION state. The protocol begins with version negotiation. The client MUST send the VERSION command to the server, specifying what version the client is operating on. If the version provided by the client is supported by the server, the server responds back to the client a successful 2-0-1 response which contains the version number that the server is operating on; the protocol moves to the USERNAME state. The server may support older client versions for backwards compatibility purposes. If the client's version is not supported, the server sends the 5-0-1 response indicating it does not support the version, and also includes the server version in this response; the connection is then terminated. Any other command sent at this state beyond VERSION and QUIT, or an unknown command, are not valid and will result in a 5-0-0 or 5-0-2 response respectively, and the connection being closed.

At the USERNAME state, the client needs to send a valid USER command with the login username. Any other command (except QUIT) is not valid at this state and the protocol will remain at the USERNAME state. Only the USER command with a 3-0-0 response will permit transition into the PASSWORD. Of important note here is that there is NO error response for an invalid username - this is important to protect from oracle attacks. The server implementation MUST return a 3-0-0 response for this command even if it is not an actual registered username.

The PASSWORD state requires the client to send the PASS command with the password for the previously provided account. If any other command is sent (except QUIT), the DFA moves back to the USERNAME state and requires restarting the entire authentication sequence. If the PASS command is sent, one of two things could happen. First, if the username and password combination is invalid (either the username or password may be wrong), then a 5-0-2 authentication failure response is sent and the DFA moves back to the USERNAME state. This response may NOT denote whether it was actually the username or password that failed. If the username and password are valid, the client is authenticated as the given username and a 2-0-2 response is sent. The DFA then moves into the ACCOUNT state.

At the ACCOUNT state, the client may now check account details and join games. A number of commands here are valid. The following commands are usable at ACCOUNT and do not cause a state transition. Sending GETBALANCE may return the user's balance in a 2-0-3 response. The UPDATEBALANCE command, if successful, will be followed by a 2-0-0 server response. GETTABLES, if successful, will be followed by a 2-1-1 response with the listing of tables per the PDU definition. ADDTABLE if successful will have the server return a 2-1-4 response with the unique ID of the newly created table. REMOVETABLE if successful will have the server return a 2-1-0 response. QUIT is also valid here as explained and will disconnect the client. The only other valid command at ACCOUNT is JOINTABLE. If the server response to JOINTABLE is 3-1-0, then the player joins the table and may bet right away. This occurs either if the player is the first client at the table (thus the first round immediately starts), or if the player joins a table during the betting phase. One important note here is that each blackjack table managed by the server has a sense of protocol state; a number of clients are associated with each table, and the server needs to manage the protocol state of each client with regard to the table that they have joined. For this reason, the server uses a timeout-based system where timeouts are used during better and during player turns to ensure that game rounds make progress and are not held up by a single player stalling. More details will be provided at the relevant states. Just know that JOINTABLE may result in a 3-1-0 if the client is the first client joining the table or if the table's internal state is accepting bets. More on this later. If the client is joining a table, is not the first person at the table and the table is NOT at the betting phase (i.e. it is some player's turn to hit or stand), then a 1-1-0 response is sent by the server, and the DFA moves to IN_PROGRESS. The client must wait until a new round of the game starts.

At IN_PROGRESS, the game is being played but the client cannot participate in the current round. The only valid commands are QUIT, GETBALANCE, and UPDATEBALANCE which operate the same way they do in ACCOUNT with the same response codes. From here on I am going to denote that every state talked about now on is a blackjack-game state, so QUIT, GETBALANCE, and UPDATEBALANCE must work at any game state described. The game states are IN_PROGRESS, ENTER_BETS, WAIT_FOR_TURN, TURN, and WAIT_FOR_DEALER. The only other command that works at every game state is LEAVETABLE, mentioned previously with the first DFA diagram. Any invalid command at any game state will result in a 5-0-0 or 5-1-0 response (general versus blackjack respectively) with no transition. The client in any of these game states may also receive 1-1-5 messages from the server. These are broadcast responses indicating events of the game. For example, a player may stand, and a 1-1-5 response is sent by the server to all other players at the table saying that "Player 1 stands". The choice of content of broadcast and exactly what gets broadcast for the other players (ignoring the dealer) relatively to a client is not enforced by the protocol. These broadcasts can be displayed in a log on the client's end but do not affect any state transitions, and thus the DFA will always remain in the same state on receiving any of these responses. Once the table is ready to start a new round, the client will receive a 3-1-0 response indicating that they should place a bet. The DFA moves to the ENTER_BETS state.

At ENTER_BETS, every player at the table should place a bet with the BET command. Once the command is sent, and a 2-1-0 response is received from the server, the player is entered into the round. The bet sent is immediately deducted from the player's balance and the DFA moves to the WAIT_FOR_TURN state. If the player's client does not send a response within the TIMEOUT window (10 seconds), the DFA moves to IN_PROGRESS; the player does not play during the current blackjack round and must wait for the next round.

At WAIT_FOR_TURN, the player waits for their turn to start. During this state, the dealer is going to distribute cards per the game rules. For blackjack, this means two cards distributed to the player's client (two 1-1-1 responses), and one face up card distributed to the dealer (another 1-1-1). If the player has a natural blackjack the player will receive a 1-1-4 response and the DFA goes to WAIT_FOR_DEALER (if the player has blackjack, they have the best hand). On the player's turn, the player's client will get a 3-1-2 response from the server, and the DFA goes to START_TURN. The turn order can be determined by the server and does not affect functionality of the protocol; for example, the players can be assigned numerical IDs and turns can go in ascending player ID order.

At TURN, the player has a number of options. First, if the player stands via STAND, the response code of 210 is sent back, and the player's turn is over. The player's hand holds their final value for the round. If DOUBLEDOWN is sent, the player doubles their original bet (assuming it fits within the balance; the double is immediately deducted), draws another card, and ends the turn. The response can vary, if successful; 1-1-1 is received by the client if a standard card is dealt (no bust, no 21), 1-1-2 is received if the player busts, 1-1-3 is received if the player doubles down without bust or reaching 21, and 1-1-6 is received if the player gets 21. The player can also HIT and get a new card; 1-1-2 and 1-1-6 are also possible responses here that denote end of turn. If no message is sent during the 30-second timeout, then the player is assumed to stand and ends the turn. All of these actions move the DFA to WAIT_FOR_DEALER (the player's turn is done). If the player HITs and does not bust or get 21, the response code is

1-1-1 for the new card, and the DFA stays at TURN.

WAIT_FOR_DEALER denotes the end of the player's turn and now the player waits for all other players at the table to finish, then for the dealer's results. The dealer will play according to the table policy (regarding hit or stand on soft 17 per the table settings). Some number of 1-1-1 messages will be sent during this time to denote what cards the dealer hits. If the dealer busts a 1-1-2 message is sent as an intermediate message. There are two ways in which the game can conclude. The 3-1-4 response here is sent once the dealer's turn is over which details the player's winnings. Once the dealer's turn is over and either of these responses are sent to the client, the next round begins, and the DFA moves back to ENTER_BETS.

To give an idea of the game, here is a sample run of how the protocol works, with states denoted through the run. Note I am just referencing the command "names" and response codes without delving into the detailed contents of each PDU, denoting some important contents as needed. These should not be interpreted as the literal PDUs, but just referencing which PDU is sent.

```
(VERSION)
client: VERSION
server: 2-0-1
(USERNAME)
client: USER <username>
server: 3-0-0
(PASSWORD)
client: PASS <password>
server: 2-0-2
(ACCOUNT)
client: JOINTABLE <id> (join table id 1)
<first player, table opened>
server: 3-1-0
(ENTER_BETS)
client: BET <bet> (suppose player bets 50)
(WAIT_FOR_TURN)
server: 1-1-1 (player gets 7H)
server: 1-1-1 (dealer gets 6D)
server: 1-1-1 (player gets 8H)
server: 3-1-2
(TURN)
client: HIT
server: 1-1-1 (player gets 4S)
server: 1-1-5 ("Player 2 has joined") (Broadcast message, another client has joined the table)
client: STAND
(WAIT_FOR_DEALER)
server: 1-1-1 (dealer gets 10C, must hit at 16)
server: 1-1-1 (dealer gets 2C, stands at 18)
server: 3-1-4 (round over, (19 vs 18) payoff is 75 if table payout is 3-2)
(ENTER_BETS)
client: LEAVETABLE
<table remains open for Player 2>
(ACCOUNT)
client: QUIT
<connection closed>
(DISCONNECTED)
```

6 EXTENSIBILITY

CBP is designed to be very extensible. A version negotiation is used on connection to check the version of the client against the server, and let the client know the server's version. The client begins by sending its version as an unsigned integer to the server via the VERSION command. The server checks this version ID against its own version to determine if it can be supported by the server. Whether the version is supported or not, the server sends back it's own version to the client in the response to let the client know what version the server is running. If the client gets a 2-0-1 response, then it is running a supported version and may use the server. If the client gets a 5-0-1 response, it is running an unsupported version, but gets the server version back in the response and the user may re-connect with another client that shares the server's protocol version. This allows for flexibility in future versions of CBP; if for whatever reason, some version of CBP breaks previous functionality, or a casino makes their own version of the protocol specific to their clients, the version negotiation can remain consistent between CBP versions and can allow only those clients that have a supported protocol version to connect to the server.

Every command PDU sent by the client starts with the same two byte header which allows for extension via more commands. Each byte is an unsigned integer that has a range of 256 values. The first byte denotes the category code. Although the first two values of 0 and 1 are already in use, the remaining 254 values are unused. More games (like slots or poker) could be added to the protocol by way of adding a new category code for the game. The DFA for the game could then be added into the overall protocol DFA with some join function accessible from the ACCOUNT state (like what blackjack does). More commands can also be added to each game; the command code allows for up to 256 unique command PDUs per game.

The response codes are also extensible. Since the second field in the response code header is the category code, then this leaves 256 values for the third field and 256 values for the first field (denoting overall reply type). This means there can be up to 256 reply codes types and up to 256 specific replies per category and type; thus a game (category) can have at most $256 \times 256 = 65536$ unique responses. Also, some responses codes are generic in terms of content; the delineation is enforced through the header and ending in LFLF, but the content of the message data can vary by server and protocol version. This provides room for even more extensibility by letting certain response codes denote grammars (in BNF) for the underlying ASCII date, which can then be parsed and used by the client. For example, the 3-1-0 response to the JOINTABLE command provides a grammar for denoting the table settings. New grammars could be made for new response codes that also need to communicate variable data. Other messages can also be used for generic purposes; the broadcast response of 1-1-5 sends events to the other players ("Player 2 has joined", "Player 2: hello world", "Player 2 hits, receives 4D", etc.), but this can be used to broadcast other information (for example, a server could send a broadcast message saying "Server closing in 20 minutes for maintenance", and the clients would already know how to parse and display it to the user).

To add more states to the DFA in future versions of CBP, new commands or response codes can be added to create edges to new states. For example, a command "STARTSLOTS" with category code 2 and command code 0 could be used from the "ACCOUNTS" state to start up a slot machine instance for the authorized user; the protocol would then move to a "SLOTS" state.

The most interesting extensibility aspect that CBP provides is the use of settings headers when creating a blackjack table. A few headers have been defined by this document, but the generic BNF grammar allows for future versions of CBP to denote extra headers. For example, a header could be added in CBP version 2 which allows for the creation of side-bets. These side-bets would be communicated to clients on joining the room, and a new command (e.g. SIDEBET) could be used to place a bet on the side-bet. Since the server maintains all game logic (clients just dictate actions), the clients would not need to know anything about how side-bets work, which would allow servers to define side-bets however they choose, as long as the side-bets that are available are communicated to the clients. Similarly, game rules beyond hitting on soft 17 could be added to change the dealer's playing pattern; this would affect what moves get broadcast and how the dealer handles cards, but would not require any change on the client end. You could even add a header that changes the value players and dealers bust at from 22 to some other number, and clients would still work with the server, because this requires no change in client PDUs or the DFA, but only a change in the server-side game logic. The client, on joining the table the first time (3-1-0 response) receives the table settings, and can disable the user from sending certain commands through the user interface if the table settings disable a command. Even if the user interface is not this sophisticated, the protocol DFA ensures that commands can only be sent at the right states if enabled.

There is one other aspect of extensibility being the card enumerations. The rank and suit are both expressed as ASCII characters, but this presents opportunity to use other ASCII characters if a modified game wanted to use

a new rank (e.g. “joker”) or a new suit. Again, since all game logic is handled by the server, the value of a hand is always communicated from the server to the client via the respective response codes. Updates to the protocol can add in new cards with defined soft and hard values, and clients would still work because the value of the hand is still communicated through the response. The cards are merely sent as auxiliary data to display back to the user. Through the use of these extensible features, variations of blackjack can be hosted on the server without necessarily requiring a client update for every new option.

7 SECURITY IMPLICATIONS

CBP itself does not provide dedicated security mechanisms (i.e. no encryption or key management) but it must use TLS (transport layer security) for all messages sent and received over the network. By using TLS, all messages are encrypted for data integrity. This also ensures non-repudiation through the key exchange done by TLS. An attacker cannot simply do packet-sniffing now to steal messages; all messages are encrypted, thereby promoting security.

The CBP server accepts all TCP connections that pass the version negotiation. Of course, it is still pretty trivial to fake a passing response to the version negotiation (since the server provides its own version in the 5-0-1 response), to make it to authentication. The protocol ensures that commands cannot be sent improperly through the design of the DFA. During the authentication sequence, any invalid command restarts the entire sequence from the beginning. Once authenticated, invalid commands (whether it be sending a category and command code that does not exist, or using a command at the wrong time) will always result in an error response and cause no state transition. Client commands that have variable length are limited in maximum size; this prevents a client from sending a continuous stream of data to stall a server. Eventually at the PDU limit for a variable length PDU (e.g. 1032 bits for CHAT), the server will stop reading and throw out the message for being invalid. The use of timeouts during blackjack also prevents users from stalling games; if no valid message that causes a transition is sent during the 30-second timeout for a given game state, the game moves on and assumes the player’s action as specified in the DFA description. Thus blackjack games are always making progress in each round regardless of what the other players do (so malicious players cannot halt games).

The use of TCP at the transport layer ensures reliable and correct message delivery, improving the quality of service for CBP. The server can be certain that it is receiving messages in the right order, and same goes for the client. Thus the CBP server knows that any commands received out of order were intentionally sent that way and can handle with an appropriate error response. This also ensures that player’s actions are sent in the proper order (sending HIT and then STAND is very different from sending STAND and then HIT), thus ensuring that the player’s intent as sent via the client is interpreted correctly at the server-side. Clients cannot try to “game” blackjack through invalid command ordering; the DFA enforces proper play of the game, and the server maintains all game logic (so clients cannot fake what their hand is, or what their value is. The server maintains this, clients just communicate actions).

The authentication provided by CBP is very basic. The client sends a USER command with a username to the server, gets a 3-0-0 response, and then sends a PASS command with a password. If the username and password are valid (the server can verify this however it likes, but ideally should store user credentials in some secure way. CBP does not define how the server implementation should go about doing this), then the client gets a 2-0-2 response and is authenticated, now able to update their balance and join tables. CBP does NOT denote a specific error response for an invalid username. The username should be verified AFTER both USER and PASS commands are sent, and the 5-0-2 authentication failure response MAY NOT specify whether the username or password was invalid. This is to prevent oracle attacks so that attackers cannot try to guess which usernames are/are not valid. The DFA as mentioned ensures that only authenticated users can update their balance or join tables, because these commands receive error responses at the authentication states (and do not allow a transition into a game). Two commands are used instead of one in case of packet sniffing; although all PDUs are encrypted, having authentication in two packets requires an attacker to get both packets and decrypt both, rather than a single one (and also verify that both packets are from the same client connection). This helps improve the security of the authentication procedure slightly.

Certain users may be allowed to use the ADDTABLE and REMOVETABLE commands once authenticated. This is up to choice of server, and could be enabled for all users or only for specific usernames (e.g. denote an admin role in a user database). CBP does not denote exactly which users can use these commands, but if a user tries to use ADDTABLE or REMOVETABLE, and is not permitted to do so, the user’s client MUST receive a 5-1-0 response to

indicate failure. CBP does not provide a notion of an admin role because of potential security concerns. Having a command to grant a user admin privileges could be very dangerous if a threat actor gets into an admin account and grants administrative privileges to other accounts. Exposing administrative management through the network is a bad idea. If interested in having dedicated admin users, this may ONLY be done at the server implementation - you cannot do this through the protocol.

Overall CBP provides very rudimentary mechanisms for security, but good quality of service and encryption is ensured through TCP and TLS. Timeouts and limited size PDUs ensure that games make progress and that clients cannot stall games. There is a user authentication procedure, designed to prevent oracle attacks, however to avoid potential security issues all notions of admin users (if needed) should be done at server implementation and cannot be done via the protocol (admins are optional, and as far as CBP cares, are not required for the protocol to work).