Network Protocol Design for Checkers

May 18, 2018

Cameron Graybill

CS 544 - Computer Networks

1. Service Description

This is the specification for a protocol to play checkers. The logic for game execution is not here, however all needed communication between the players and the game are. This system controls how users log in using a username and password, are given a match from a user queue, and interact with the server to play the game.

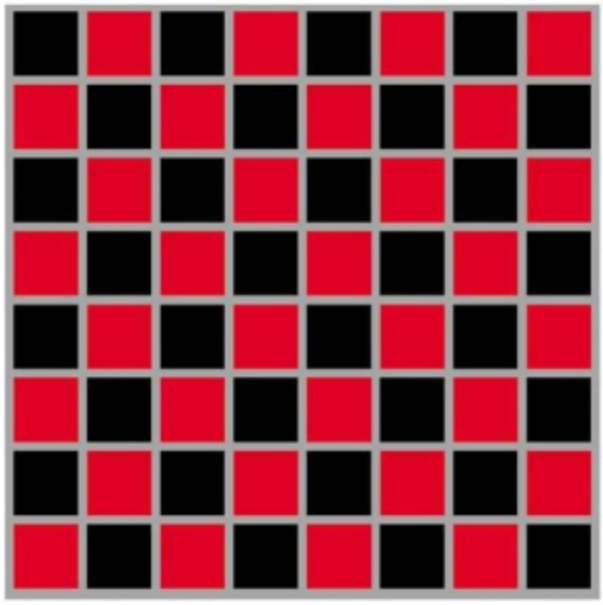


Figure 1: Example Checker Board
Source: Amazon.com

Checkers is a simple game played on a board that is 8 by 8 squares, alternating red and black colors. The game is played between two players, each with twelve pieces of their respective color (red or black). Players take turns moving pieces, capturing opponent's pieces. The game is played until one player has no more pieces. The logic for the game and it's representation in the protocol will be explained further in this paper.

Each user has a rating that is assigned based on the amount of games that that user has won and lost. After a game is finished, both user's ratings are updated. That rating is then used to match players with their next opponent, pairing the two closest in rating users.

There are currently no optional things in this protocol, however there is a version negotiation mechanism so a future implementation and version of this protocol could add optional features with more negotiation states.

This service will be running over a TCP connection that has been encrypted using TLS. The TCP socket for the server should be listening on port 8864. These two tools together give us very high reliability and a good amount of encryption to keep user information safe.

2. PDU Descriptions

2.1 General Notes

There are two common things from all of the PDUs for this protocol. Each message is required to have a 'message_type' field and is required to be a predefined length. The fields in the message come in order top to bottom in the tables below, making sure that the

‘message_type’ field is always first. The messages are designed this way to make both encoding and decoding the messages easier. A simple parser for these messages could work as follows:

1. Check the first byte of the message for the message type
2. Look up the constant size of that message
3. Read the message size - 1 bytes from the socket
4. Pass the read data off to a decoder to use the message based on current protocol state

2.2 Message Delineation

Because all messages use a ‘message_type’ field, there is no message delineation. The clients and server will read and write exactly the right amount of bytes.

2.3 Data Types

2.3.1 Numbers

All numbers in this protocol are encoded little endian (least significant byte first). All numerical values are unsigned. Integers will be either 1 or 4 bytes in length.

2.3.2 Strings

All string values in this protocol will be encoded in ascii. Each character will be 1 byte, and the string will be filled in to the right with null values. All strings are also set to be length 16 so 16 bytes. If a user had the username “username”, then that would be stored in a message as “username\0\0\0\0\0\0\0\0”. When parsing a string, the parser may stop after finding the first null character, and then jump to the end of the 16 bytes to start parsing the next field.

2.3.3 Enumerations

All enumerations are represented by 8 bit integers, and are defined in their specific message definitions.

2.3.4 Custom Types

Move

The first custom type that is used by this protocol is a move. A move is a representation of a piece being moved on the board. Moves are encoded as a single byte integer, as shown below. The X and Y Positions are decoded as 3 bit integers (allowing values 0-7). The X and Y Direction values signify whether the piece is moving in the positive or negative direction on that axis. A 1 represents positive direction, 0 represents negative direction.

Move		
Field Name	Size (bits)	Description
X Position	3	The X location on the checkerboard of the piece to move
Y Position	3	The Y location on the checkerboard of the piece to move
X Direction	1	The X direction to move the piece on the checkerboard
Y Direction	1	The Y direction to move the piece on the checkerboard

Board

The second custom data type that is used by this protocol is a board. A board is a representation of the current state of the board. A board is encoded as 24 bytes of data, or 64 sets of three bits. Each set of three bits represents a square on the board. The data is stored such that the first three bits are the values for the location of the board where both X and Y are 0. The next value is when X is 1 and Y is 0, etcetera. The three bits of each board location represent if there is a piece there, which player owns the piece and if the piece is promoted. It is encoded as shown below. This data type is only ever sent from the server to the client, and it is represented as an array of 24 unsigned ints, but any set of 24 bytes would work.

Board Location		
Field Name	Size (bits)	Description
Used	1	If the position has a piece in it
Promoted	1	If the piece is promoted
Owner	1	If the receiving user is the owner of the piece

2.3.4 Message Definitions

Connect

Connect			
Field Name	Size (bytes)	Data Type	Description (or value)
message_type	1	Unsigned Integer	0x01
version	1	Unsigned Integer	The version of the protocol to respond with
username	16	String	The connecting user's username
password	16	String	The connecting user's password

The Connect message is sent from the client to the server in order to log in to the server. This is the first message sent once the TCP connection has been established. This message is also used to check that the client and the server support the same version of the protocol.

Invalid Login

Invalid Login			
Field Name	Size (bytes)	Data Type	Description (or value)
message_type	1	Unsigned Integer	0x02
reason	1	Enum	The reason the user was not authenticated.

The Invalid Login message is sent from the server to the client in response to the client giving the server a bad login. The 'reason' enum can be the value 0x00 for "Account Does Not Exist" or 0x01 for "Invalid Password".

Invalid Version

Invalid Version			
Field Name	Size (bytes)	Data Type	Description (or value)
message_type	1	Unsigned Integer	0x0D
highest_supported_version	1	Unsigned Integer	The highest version this server supports
lowest_supported_version	1	Unsigned Integer	The lowest version this server supports

The Invalid Version message is used for version negotiation. This is sent from the server to the client when the client attempts to connect with a version that the server does not support. The server responds with the highest version that it supports and the lowest version that it supports, indicating that the client should then attempt to log in again using a protocol version in that range.

Queue Position

Queue Position			
Field Name	Size (bytes)	Data Type	Description (or value)
message_type	1	Unsigned Integer	0x03
queue_size	4	Unsigned Integer	The amount of players in the queue
queue_pos	4	Unsigned Integer	The current position of the client in the queue
rating	4	Unsigned Integer	The client's current rating

The Queue Position message is sent from the server to the client on a one second interval while the client is waiting for a game to start. The client can then use this data to show the user their position in the queue and estimate how long it will be until they are placed in a game.

Game Start

Game Start			
Field Name	Size (bytes)	Data Type	Description (or value)
message_type	1	Unsigned Integer	0x04
opponent_name	16	String	The username of the client's opponent
opponent_rating	4	Unsigned Integer	The client's rating

The Game Start message is sent from the client to the server when a match is made and the client needs to start playing the game. It includes all of the metadata about this match that the user needs to know, specifically the user's opponent's name and rating. This information is for the user's use only any does not need to be preserved for the protocol.

Your Turn

Your Turn			
Field Name	Size (bytes)	Data Type	Description (or value)
message_type	1	Unsigned Integer	0x05
last_move	1	Custom (Move)	The last move that was made
board	24	Custom (Board)	The current state of the board

The Your Turn message is sent from the server to the client when it is the client's turn to make a move. The message includes the last move that was made and the current board (after the last move was applied) so that the client can update it's representation of the board.

Make Move

Make Move			
Field Name	Size (bytes)	Data Type	Description (or value)
message_type	1	Unsigned Integer	0x06
move	1	Custom (Move)	The move the client wants to make

The Make Move message is sent from the client to the server when it is the client's turn to make a move. This message consists of just the message_type and the move that the client would like to make.

Compulsory Move

Compulsory Move			
Field Name	Size (bytes)	Data Type	Description (or value)
message_type	1	Unsigned Integer	0x07
move	1	Custom (Move)	The move that was made
board	24	Custom (Board)	The new board state

The Compulsory Move message is sent from the server to all clients when a client is forced to make a certain move. This message does not affect the state of the protocol, however it does effect the state of the game so it must be sent to the clients in order for them to keep their board states synchronized with the server.

Invalid Move

Invalid Move			
Field Name	Size (bytes)	Data Type	Description (or value)
message_type	1	Unsigned Integer	0x08
move	1	Custom (Move)	The move that was invalid
board	24	Custom (Board)	The current board state

The Invalid Move message is sent from the server to a client when the client attempted to make an invalid move with the Make Move message.

Opponent Disconnect

Opponent Disconnect			
Field Name	Size (bytes)	Data Type	Description (or value)
message_type	1	Unsigned Integer	0x09

The Opponent Disconnect message is sent from the server to the client when the client's opponent has disconnected from the server. This transitions the protocol state from wherever it is to the "Game End" state.

Game Over

Game Over			
Field Name	Size (bytes)	Data Type	Description (or value)
message_type	1	Unsigned Integer	0x0A
you_won	1	Unsigned Integer	0xFF if the client won, 0x00 if the client lost
new_rating	4	Unsigned Integer	The client's new rating
old_rating	4	Unsigned Integer	The client's old rating
winning_move	1	Custom (Move)	The last move that was taken
board	24	Custom (Board)	The end board state

The Game Over message is sent from the server to the clients when the client's game has ended.

ReQueue

ReQueue			
Field Name	Size (bytes)	Data Type	Description (or value)
message_type	1	Unsigned Integer	0x0B

The ReQueue message is sent from the client to the server when the client decides to rejoin the queue after finishing a game.

Log Out

Log Out			
Field Name	Size (bytes)	Data Type	Description (or value)
message_type	1	Unsigned Integer	0x0C

The Log Out message can be sent from the client to the server at any time in order to forfeit their current match (if they are in one) and log out of the service.

3. DFA and States

3.1 State Descriptions

This protocol has 5 total states. The transitions between the states are described by the messages that are in section 2.3.4.

3.1.1 Unauthenticated

This is the start state and the end state. When a TCP connection is first established, the protocol is in this state. This state is also used for version negotiation and authentication checking.

3.1.2 In Queue

In Queue is the state that a client is put in once they are successfully authenticated with the server. The clients in the “In Queue” state are waiting for a match to be made for them. While in this state, a message is sent from the server to the client every second to update where the client thinks it is in the queue.

3.1.3 Processing Game State

Processing Game State is the state the protocol is in while the server is processing actions. There are transitions to and from Processing Game State based on some game related actions, however these actions also need to be network events so they count as transitions.

3.1.4 User Move

User Move is the state where clients make inputs to play the game. Usually the state will switch between User Move and Processing Game State for many iterations until one client has won the game.

3.1.5 Game End

Game End is the state that is transitioned to when one client wins the game. This can happen various different ways. At this point the clients can choose to log out and disconnect or to rejoin the queue to play another game.

3.2 DFA

In the DFA, “C -> S” means the message is sent from the client to the server and “S -> C” means the message is sent from the server to one or more clients. See Appendix A for a larger version of the DFA that is easier to read.

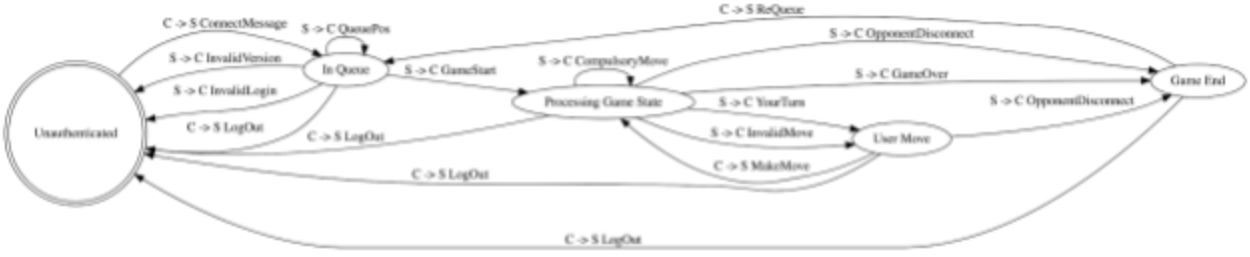


Figure 2: DFA Describing this protocol

4. Extensibility

4.1 Versioning

This protocol is extensible through versioning. When the client first connects to the server, it must provide the version of the protocol that it is connecting with. If the server does not support the version of the protocol that the client connected with then the server responds with the range of versions it supports. The client can then reconnect using that version. After the version negotiation, the DFA and control flow of the protocol would change based on what version it is using, so a later version of the protocol could add more optional features.

4.2 Message Types

The protocol is also extensible by adding more messages, there could be up to 65535 total messages per version, so there is plenty of space to add more functionality and protocol states if more features would be added.

4.3 Error Codes

The error message that rejects a user from logging in uses an Enumeration to represent an error code. This is an opportunity for extensibility. If there are added reasons for a user not being able to connect, like maybe a user being banned or the server being full, they could be added as error codes here.

5. Security Implications

5.1 Encryption

This protocol is implemented on top of an SSL secured TCP connection. This ensures that the passwords the users send are very hard to decrypt. It is up to the server implementation to make sure that passwords are stored on the server in a secure way as well.

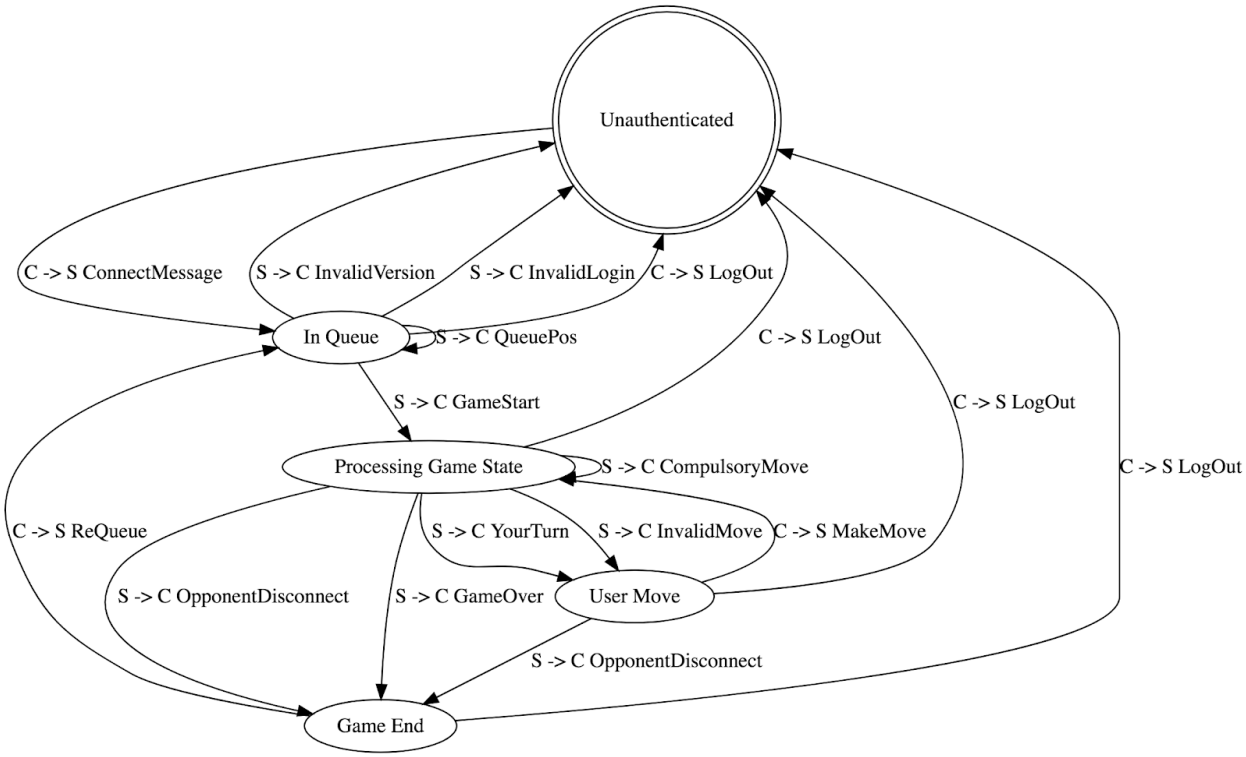
5.2 Access Control

Everybody can connect to the server with a secured TCP connection, however the user must have a valid username and password to proceed past the first two states of the protocol. A user account can be created by an outside source.

5.3 Fixed Sizes and Message Types

The usage of fixed length messages and and fields adds security to the protocol because there are no variable length values that could cause a buffer overflow. Using message types also adds security to the protocol because if the server or client receives a message it is not expecting for the current state, it can report an error or a warning and close the connection. Because of how TCP delivers messages they should always come in order, so getting an out of order message probably means the other side of the connection has the protocol implemented the wrong way or is trying to exploit something.

Appendix A


Unauthenticated
C -> S ConnectMessage
S -> C InvalidVersion
S -> C InvalidLogin
C -> S LogOut
In Queue
S -> C QueuePos
C -> S LogOut
S -> C GameStart
C -> S LogOut
Processing Game State
S -> C CompulsoryMove
C -> S LogOut
C -> S ReQueue
S -> C YourTurn
S -> C InvalidMove
C -> S MakeMove
S -> C OpponentDisconnect
S -> C GameOver
User Move
S -> C OpponentDisconnect
Game End