Laura Maisenhelder
5/11/2019
CS544

Protocol Design

Service Description

This protocol is an application-layer protocol which will allow a client to connect to a server to play a game against the server. Client users will provide a username and password to log in. If the server does not recognize the user, it will allow them to try to enter the credentials again. If they are recognized, then the server will ask if they are ready to begin the game, and if they indicate that they are ready then that response is sent to the server to indicate so and the game begins. If they are not ready the client will notify the server and disconnect. At the end of each game, the server will let the client know who won and ask again if the client is ready to play; a positive answer starts another game, a negative answer disconnects the client.

This will follow a similar format as the Stop and Wait protocol because only one message is sent at a time. After each side sends a message to the other, it starts a two-minute timer. If the sender does not receive a response from the other side before the timer expires, it resends the message and restarts the timer. If there is still no response after the second attempt, the game will be disconnected. This way the game does not stay open if the client or server is no longer active.

This protocol will be demonstrated using a TicTacToe game, although there will be an option to select other games if needed in the future as long as they are similar style board games. The board will be a nine-character String and the client will use X's to mark their spaces while the server will use O's. The client must print the String in such a way as to create three rows of three characters each, so the game can be easily interpreted by the user. After each move, the game board will be sent back to the opposing side for the next move along with a code to indicate the status of the game. The board will print to the client console and the user will input numbers to indicate x/y coordinates of the next space they want to play on the game board or Y/N to indicate if they wish to start a new game.

PDU and Messaging

The client will connect to the server on port 1030 using TCP. Messages will have a header with a field for the three-digit control code, a field for a single letter option choice to allow choice of different games in the future (for now T will indicate TicTacToe), a field for a single letter to indicate the version number of the protocol (A = version 1, B = version 2, and so on), followed by the data of the game board nine character String. The message will use ASCII characters in UTF-8 encoding. All client and server codes will be three-digit numbers as defined in the list below. Some messages may not have any data or option filled out (such as messages asking if the user is ready to begin, or errors, or any other messages outside the 300 level of codes indicated below), but they will always have a three-digit code.

Three digits for the control code x one 8-bit byte per ASCII character will require a 24-bit control code field. A single character option indicator will require an 8-bit option field. A single character version indicator will require an 8-bit version field. This will complete the 40-bit header, after which the 72-bit (9 character ASCII string x one 8-bit byte per ASCII character) String game board data will follow, and the PDU will be completed by an end of message

indicator which will comprise another 24 bits because it will be a three letter ASCII character code, EOM. My particular implementation will be done using the Java programming language, where a string is a sequence of characters that has a property of a fixed length, so there is no specification of null-terminated strings for my particular implementation because it will be done using Java.

The PDU will be structured as follows:

<table>
<tr><td colspan="2">24-bit control code field</td></tr>
<tr><td>8-bit option field</td><td>8-bit version field</td></tr>
<tr><td colspan="2">72-bit data field</td></tr>
<tr><td colspan="2">24-bit End of Message indicator</td></tr>
</table>

Codes in the 100's will be for user authentication. The 200's will be for whether the game will be initializing or not. The 300's will be for game status during play, these codes will be the only ones that are followed by the data of the game board. The 400's will be for an error on the client side. The 500's will be for an error on the server side.

To begin, the client will attempt to connect to the server by sending a message including code 100, a version indicator, an option indicator for the game, and user credentials to log in. The server will receive the credentials (an ASCII String composed of username, comma, password, comma, protocol version, comma, game choice, followed by End of Message indicator) and attempt to authorize them. If the user is not verified, server will send code 175 to indicate that the username or password was incorrect and to try again. If the user is verified, and if the server can support the version requested, and it can provide the game requested, it will send code 150 and a return of the version number and game desired to indicate that they are authorized and that the protocol version and game requested are supported. The client will display a message to the user to ask if they are ready to begin. If the server is not able to support the version requested by the client, the server will send code 115 to indicate this and the client will disconnect. If the server is not able to support the game requested by the client it will send code 120 to indicate this and the client will disconnect.

If the user accepts, the client will send code 200 to the server to indicate that the game should begin. If the user declines, code 250 is sent to the server to indicate that the client will be disconnecting. This will be useful later when, after one game completes, the server can ask the client if they wish to play again, so this exchange can be used multiple times.

If the code 200 was received, the server will initialize the game board and send it to the client with response code 300 to indicate that the game has begun, it will also begin its message timer for two minutes. The client will input their move (again an ASCII String, this time it will only contain two characters, such as A1 or B3, followed by the End of Message indicator) and send the board back to the server along with code 350. This tells the server to check for a win and if no win, make its own move. The client will also ping the server if its own two-minute timer expires. If there is no win, the server will make its move and send the board back to the client with code 325 to let the client know it is their turn again while the server waits and starts its own timer again. This cycle will repeat until either there is a winner or the board is full and no more moves can be made.

If the server wins, it will send code 375, if the client wins the server will send code 380, if there is a stalemate and no one won it will send 385. The client will then inform the user of the outcome based on the code it received and ask if the client wants to play again, which would prompt a return code of either 200 (to start a new game) or 250 (to stop) to the server.

In case connection is lost unexpectedly, or incase a player wishes to quit before the game is over, the server and client will both have timers that are initiated when they send any message, and after a certain interval will ping the other side to see if they are still there. The server will send code 400 to ask the client if the game is still continuing, and the client will send code 500 to ask the server if the game is still continuing. If the server gets no answer, code 450 displays on the server side to indicate client failure. The client gets no response from the server, code 550 will display on the client side to indicate server failure and the client socket will close.

List of client and server response codes **(if data is to be sent with the message it is defined in parenthesis):** and again all messages and data will be ASCII characters in UTF-8 encoding

100 level for verification and negotiation

100 incoming client (sending username, password, protocol version, game choice option)

115 protocol version not supported

120 game requested not supported

125 client credentials received by server, verifying

150 client verified, game may begin, ask client if they are ready

175 client not verified, enter credentials again

200 level for game initialization

200 client ready, server initialize game

250 client wishes to disconnect and the client socket will be closed

300 level for game play

300 server delivers board and game has begun (game board sent as String)

325 game board sent by server, server waiting (game board sent as String)

350 game board sent by client, client waiting (game board sent as String)

375 game over, server won

380 game over, client won

385 game over, stalemate

400 level client-side error

400 server asking client if they are still playing

450 client failure, ping got no response

500 level server-side error

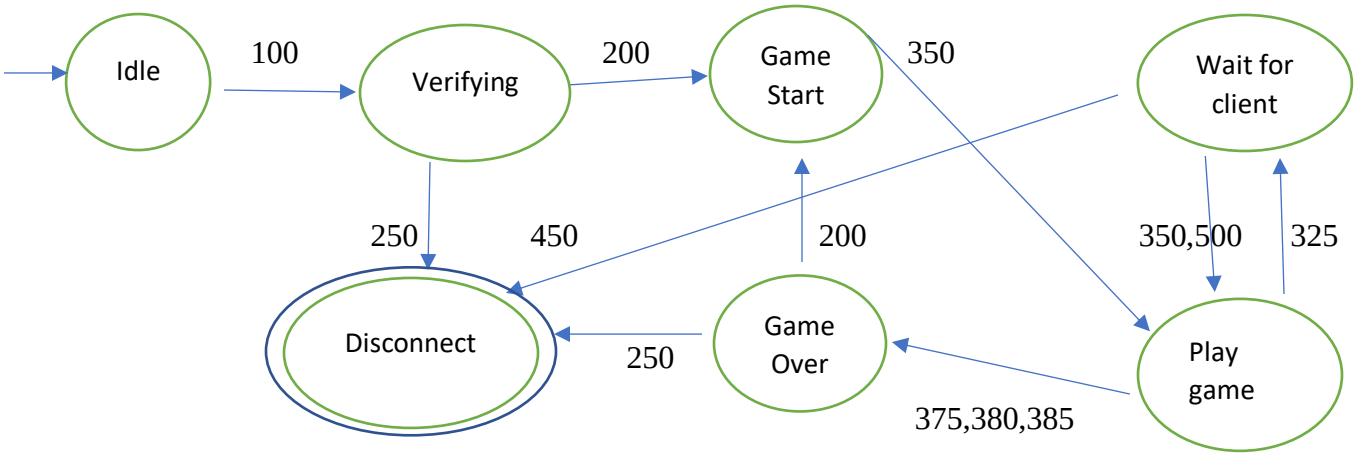
500 client asking server if they are still playing

550 server failure, ping got no response

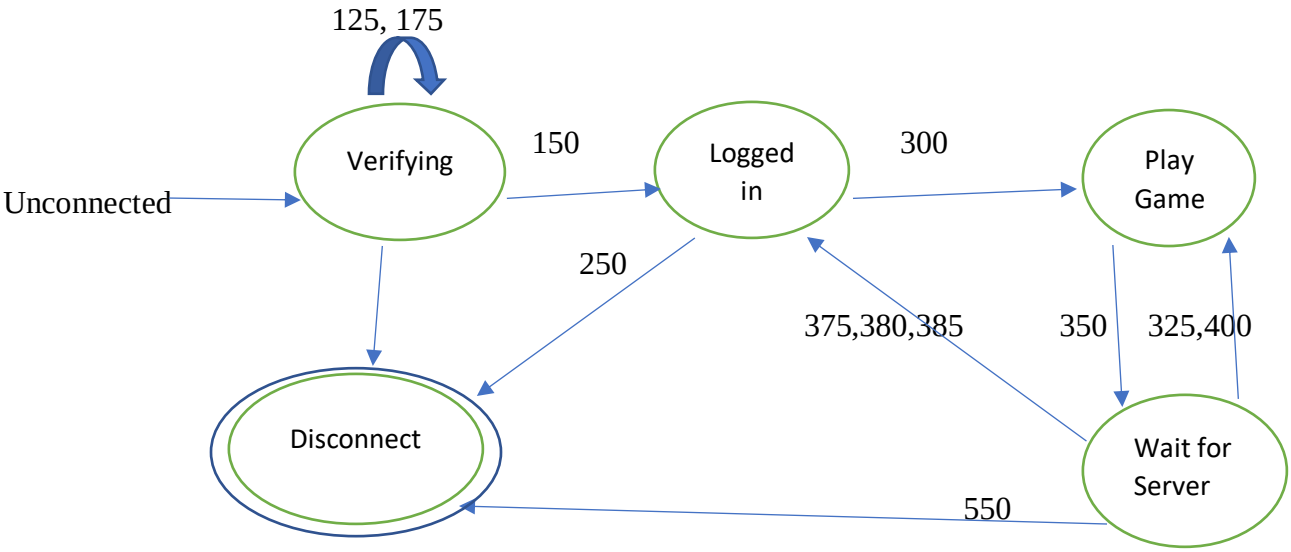
DFA

The diagrams below will indicate, for either the server or client (as specified), the states and transitions between them caused by the codes written next to the arrow, which would accompany a message from the opposing side and be an instruction to the receiver as to what protocol state to move to next. For example, in the Server DFA below, the server starts in the idle state and when it receives code 100, moves to the verifying state. Multiple codes separated by a comma but next to the same line indicate that in different situations, any one of those codes would cause the same transition to happen.

Server DFA:



Client DFA:



Extensibility

This protocol could be extended in the future to include additional games. The options field contains one letter, and only one letter in the alphabet has been taken so far; the T to indicate TicTacToe. This leaves room for up to twenty-five additional board games to utilize this same protocol, for example checkers or chess. The PDU could also be re-defined to hold a larger game board, a later version of the protocol could perhaps have this expansion, it would just need to define the byte size necessary based on the size of the game board.

The command codes are three digits long which also leaves room for expansion. The format of 100's vs 200's codes being grouped together for different purposes helps keep it organized and gives a clear guide for which types of statements would go in which level of codes, for example more game commands would be in the 300's. If new codes were added, then the DFA could also be extended to include new states if necessary.

Security

The service will require a username and password to log in, for testing purposes this will be username Bob with password 1234. Currently, the protocol does not have an option to create a new username and password, but if time allows, I might add that option before the final submission. This will also be run over TLS to provide encryption.

Reliability

The protocol sends only one message at a time and waits for a response with a control code before proceeding. Each message that is sent starts a timer on the sender's side for two minutes after which the message will be resent if no response has been received. If the second message does not receive any reply, then the game is abandoned and the connection is closed. This way if one side stops responding for whatever reason, for example if the client user walked away from their machine or closed out of the game, the other side gets an indication of that. For more critical situations such as a transportation or medical application, this method of response in abandoning the connection so quickly would not be appropriate but this is for a small board game just to know if either side has abandoned the game so it doesn't leave the client socket open unnecessarily or occupy the server for extra time. This server is designed to handle one client at a time, so it would be beneficial to close a connection that is not actively being used so that a new one could be opened. TCP will handle tracking and resending of packets if needed.