Nikita Patel
February 11, 2020
CS 544 – Computer Networks
Network Protocol Design

The Game Selection Protocol

The Game Selection Protocol

The Game Selection Protocol is a flexible application layer protocol that enables two player game play across a spectrum of gaming applications. The client-server and text-based protocol ensures both reliable as well as secure game play through its design.

Service Description

The objective of Game Selection Protocol is to allow any applications capable of representing its game state deterministically and as a series of 8-bit ASCII characters, to enable secure gameplay amongst two opponents. A client begins by connecting to a GSP server, establishing a unique username and a password for authentication purposes, and then using that username and password to log in. Once the client is authenticated, he or she is registered user in the gaming environment which means they can initiate, or be asked to participate in, a game at any moment. The GSP server plays a very large role as it oversees authentication as well as keeps a list of all active participants, their hostnames, and all the games they can play.

Once a player decides to initiate gameplay, they can do so in one of two ways. The user can either request a list of all active users from the GSP server or specify the username of their desired opponent directly along with the game they would like to play. The specified opponent username and chosen game are validated and checked on the GSP server, and if the opponent exists and they are capable of partaking in the specified game, the request can go through. The chosen opponent can either accept or deny the invitation to play. If the game is accepted, the requesting player can start the game at any point. Once a game begins, game state as well as other game information is passed between opponents using the server as an intermediary.

Message Definition

Game Selection Protocol messages are composed of a series of 8-bit ASCII characters with no maximum placed in message length. Each message is composed of four parts – version number, username, command and arguments.

1. Version Number – the version number will be specified as a series of 4 ASCII 8-bit characters indicating GSP version allowing clients and servers alike to know how to interpret the messages across the network.
2. Username - unique identifier that the client has used to register and authenticate itself with the GSP server. The username can consist of up to 16 8-bit ASCII characters (128 bit).
3. Command - 4 ASCII 8-bit characters out of the specified GSP commands
4. Arguments - there is no maximum number of arguments that a command can require or suggest and there is no length limit to arguments either.

The message overall is somewhat ambiguous in nature outside of these four requirements, especially in length cutoffs. This is done intentionally since GSP is meant to be a multi-use protocol, to be used across applications, and the requirements of the applications can never truly be known. Also, since the protocol is meant to be used by many different games that can vary greatly in complexity of state representation, it would not have been wise to impose a limit on the overall message length. Messages can and will be represented as shown below.

```
<message> ::= <version> <sp> <username> <sp> <command> <sp> <arguments> <crlf>
<version> ::= <char><char><char><char>
<username> ::= <string>
<password> ::= <string>
<command> ::= <char><char><char><char>
<arguments> ::= <string> | < arguments > <sp> <string>
<hostname> ::= see RFC 952 [DNS:4] for details on allowed hostnames
<server-name> ::= <hostname>
<opponent-username> ::= <username>
<games> ::= < game-name> | < game-name>,<games>
<game-name> ::= <string>
<score> ::= <string> ':' <string>
<game-state> ::= <string>
<winner> ::= <username>
<string> ::= <char> | <char><string>
<char> ::= any of the 128 ASCII 8-bit characters excluding <CR> and <LF>
<sp>
<crlf>
```

As is seen in the message definitions below and the BNF representation above, since a fixed length requirement is not specified, a clear delimiter is required to indicate the end of a message and in the case of GSP that delimiter is CR;LF. Anything in a message after a CR;LF will be ignored and if the delimiter fails to appear in the message, the message will be ignored silently.

Username Message
USER <SP> <username> <SP> <hostname> <SP> <server-name> <CRLF>
The USER command is used at the beginning of every communication. Here a user will specify a unique identifier that the server will use to identify a client and store its information in a database. The username also allows for opponents to identify one another during gameplay serving as an alias so an opponent does not need to know the hostname of the opponent. For this reason, it is required that the username be unique.
Since the user command is how a client 'signs in' it is required that both the clients hostname, as well as the hostname of the GSP server (the server-name) be specified in the initial message allowing the client to connect to the GSP server and establish its identity.
Replies: ACK_USR_EXISTS, NAK_USER_DOES_NOT_EXIST
Errors: ERR_NONUNIQUE_USER, ERR_INVALID_SERVERNAME

Password Message
PASS <SP><username><SP><password> <CRLF>

The PASS command is used to set a user password on the server for the purpose of future authentication. The password can be of any length and can be any combination of 8-bit ASCII characters. This is also the means through which a user creates an account on the GSP server.
Replies: ACK_ACCT_CREATED, NAK_ACCT_CREATION_FAILED

Log In Message
LOGN <SP> <username> <SP> <password> [<games>]
A client, once a username and password have already been set, must log into the GSP server and authenticate its identity. On valid login and server authentication, the player will be logged in. On login is also when a client can specify any additions to its list of available games for the server to store (optional). The list of games will be a combination of game-names separated by a comma and no spaces. Game names must also be specified without spaces eg. tic_tac_toe.
Replies: ACK_LOGGED_IN, NAK_LOG_IN_FAILED

Get Potential Opponents Message
GPO <CRLF>
This command allows a user to get a list of all logged in users in the protocol by username, allowing a client to select a potential opponent. This command can only be sent outside of gameplay, if sent during a game the command will be quietly ignored.
Replies: List of possible opponent's usernames separated by commas with no spaces in between.
Replies: NAK_NO_OPPONENTS_FOUND, NAK_COULD_NOT_RETREIVE_OPPONENTS

Get Possible Games Message
GPG <SP> <opponent-username> <CRLF>
By specifying an opponent username, a client can then get the list of potential games that the opponent is able to play. The server keeps track of such information and returns the full list to the requesting client. This command can only be sent when a user is logged in, if sent during a game the command will be quietly ignored.
Replies: List of possible games to initiate usernames separated by commas with no spaces in between, NAK_NO_GAMES_SPECIFIED

Request Game Play Message
REQ <SP> <opponent-username> <SP> <game-name> <CRLF>
Client can send a request to play a game with an opponent by specifying a game and opponent username. The server checks for validity of the username against its internal database and ensures the opponent is capable of partaking in the specified game. If both conditions are satisfied, the potential opponent is sent the request. This command can only be sent when logged in to a user that is also logged in otherwise the command will be quietly ignored.
Replies: ACK_REQ_SENT, ACK_REQ_ACCEPTED, ACK_REQ_DENIED, NAK_REQ_FAILED, NAK_INVALID_OPPONENT

Accept Message
ACPT <SP> <opponent-username> <SP> <game-name> <CRLF>

Replies: ACK_ACPT_SENT, NAK_USER_NO_LONGER_PARTICIPATING

Deny Message
DENY <SP> <opponent-username><SP><game-name> <CRLF>
Replies: ACK_DENY_SENT

Start Game Message
STRT <SP> <opponent-username><SP><game-name> <SP> <score> <SP> <checksum> <SP> <game-state> <CRLF>
Games are started using the STRT command followed by the game name, initial score, a checksum of the games state followed by the initial state of the game. Once this command is sent and both clients are participating in gameplay. This command can only be sent outside of gameplay, if sent during a game the command will be quietly ignored.
Replies: ACK_START_SENT, ACK_START_RECEIVED, NAK_START_FAILED, NAK_USER_NO_LONGER_PARTICIPATING

Make Move Message
MV <SP> <opponent-username><SP><game-name> <SP> <score> <SP> <checksum> <SP> <game-state> <CRLF>
During game play, a client may make a move via the MV command. The move will include an updated score as well as an updated representation of the game state and a checksum representing the new state for the next user to use to update their known state. This command is only valid when the user is participating in a game, if executed outside this state, the message will be silently ignored.
Replies: ACK_MOVE_SENT, ACK_MOVE_RECEIVED, NAK_MOVE_FAILED_TO_SEND, NAK_USER_NO_LONGER_PARTICIPATING

End Game Message
END <SP> <opponent-username><SP><game-name> <SP> <winner><SP> <checksum> <CRLF>
End command is sent by the application at the natural completion of a game, whether it end in a winner or a tie. This command is only valid when a user is participating in a game, if executed outside this state, the message will be silently ignored.
Replies: ACK_END_SENT, ACK_END_RECEIVED, NAK_END_FAILED_TO_SEND

Quit Message
QUIT <SP> <opponent-username><CRLF>
The QUIT command is used when a user wishes to no longer participate in the game it is currently participating in. It can also be used when a client has accepted to play a game with an opponent but no longer wishes to do so and would like to get back to an active state so it can receive and send requests to others. This command can be sent when a client is not in gameplay but it has committed to game play with an opponent, otherwise the command will be ignored quietly.
Replies: ACK_QUIT_SUCCESSFUL, ACK_OPPONENT_QUIT, NAK_QUIT_FAILED

Disconnect Message
DCNT <CRLF>
DSCT disconnects the client from the GSP server logging it out. Once the DCNT command has been sent, the client is no longer considered an active participant, they are inactive and cannot participate or be invited to games. This command can only be sent while a client is in a logged in state, if sent during a game the command will be quietly ignored.
Replies: ACK_DSCT_RECEIVED, NAK_DSCT_FAILED

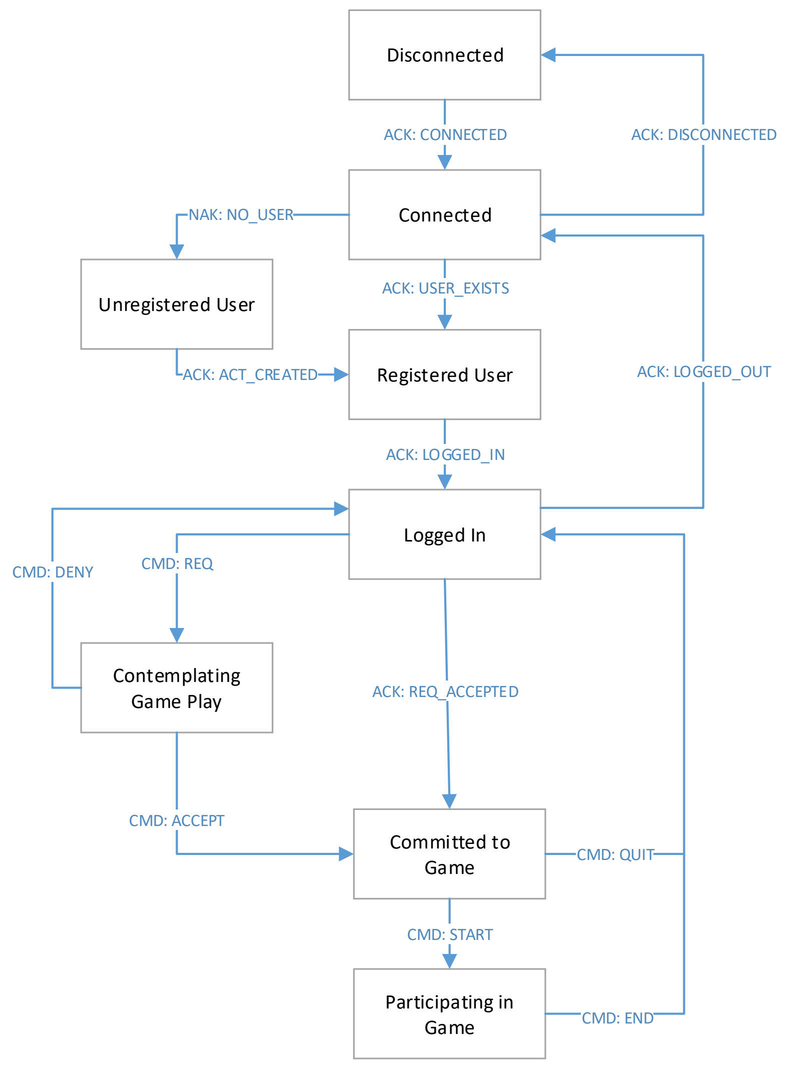
In many situations, commands sent in the wrong state are silently ignored. This was a deliberate decision to aid in the security of the protocol to keep malicious actors from knowing the exact state. This is also, however, mostly to reduce overall chattiness and thus improve scalability.

As mentioned previously, GSP relies on TCP/IP to provide an additional layer of reliability for gaming applications. However, it could be possible that future games may have less desire for reliability and more so for speed, thus port 1058 was chosen as it is an unassigned port for both UDP and TCP. This allows for future versions of the application layer protocol to be flexible.

The primary purpose of the Game Selection Protocol is to enable secure game play across a network amongst two opponents. It is incumbent on applications to provide an environment through which users can play these games. The application must take care of scoring, incrementing and decrementing points based on 'moves', determining if a move is valid, if a move was taken at the correct time, and most of all game state representation. GSP is not built to handle such functions.

Deterministic Finite Automata
There are seven total states that a client can be in as part of the Game Selection Protocol. The client will always start off in the Disconnected state until the moment it successfully connects the GSP server, Disconnected is the starting state/ initial entry point. The client once connected is in the Connected state which is where it stays until it is determined whether or not that client/user already has an account within GSP. If an account exists, they are considered a registered user. If one does not exist, the client is an unregistered user until an account is create after which they are in the Registered User state. On successful login the user can then play a more active role in the protocol, i.e. play and initiate games. If a user sends out a game request to another user, their state remains unchanged, they are still 'Logged In'. The idea behind this was that a user could send out multiple game requests within this state and be able to wait for responses and choose one of the responding opponents. If the user were to be considered, say 'Committed' upon sending out a request, this would not have been as easy to do. An opposing user, on receiving a game request is then in a state where it is 'Contemplating Game Play'. This was done so that any user can only consider a single request at a time. If the request is accepted, the opposing user is then committing to the game and if it is not, it goes back into a 'Logged In' state. Then, once a user starts the game, both users enter the

'Participating in Game State'. Once a game is over or ended, users return to the 'Logged In' state and can then go back to being disconnected via the DSCT command.



Extensibility

It is extremely important for any protocol, regardless of the layer at which it function or the functionality it aims to enable, to be extensible by design. With the rapid growth of technology, for a protocol to remain relevant and useful, it must be able to iterate and expand with the internet. There are many deliberate design decisions within the Game Selection Protocol that were made to allow extensibility.

One reason why HTTP has remained so relevant is because there is no real message definition that its messages need to adhere to, thus it was easy for that message to evolve over time with the internet. While it would be both impractical and unsecure for GSP to have no real message definition, the loose nature of the definition allows it to be inherently extensible. To begin with, there are very few limits on message length. As technology progresses and computers get faster and more efficient, the number of bits they can process at one time will also go up allowing more and more data to be put through a protocol at once and be processed. It is also important to note that GSP is not made for any particular gaming application, instead it is meant for a variety of gaming applications. This means that it is impossible to know how game state will be represented by an application so no real criteria can be enforced nor should it be enforced. The lack of a limit to the number of bits allow in the message will enable GSP to grow, both literally and figuratively with time.

The most obvious way in which extensibility is seen in GSP is through the user of version numbers. The use of a versioning is going to allow for more drastic difference in iterations of GSP. It could allow the next version of GSP messaging to look completely different than the one specified here and still work on all machines that can use older versions of the protocol. Version numbers allow the clients and servers interacting with the message to know exactly how to interpret and decode it and enable security at the same time. It is also assumed that versioning will be used in the future to expand GSP from being a two-player game protocol to potentially allowing for multiplayer game play.

While this iteration of GSP is made chose to use TCP for its implementation, the intent is to make the protocol network layer protocol agnostic allowing it to hopefully be used over both TCP and UDP. This would allow the Game Selection Protocol to be potentially be used for more modern-day video games that are more in line with streaming than that traditional board games that are turn based. Having GSP operable over both UDP and TCP would greatly expand the number and kinds of applications capable of using it. As technology advances, gaming environments change, and as more advanced network layer protocols are developed it is even more important to ensure GSP is completely agnostic to the network layer.

Security

For the Game Selection Protocol to enable secure and reliable game play, several measures were taken. The first and most obvious of those steps was basic authentication of users. A user is required to sign in with a username and passcode that is tracked by the GSP server with which it needs to log into. Having a log in process allows for an additional layer of trust in the protocol because clients can know that they are playing against who the opponent says he or she is. On top of that encryption will be built into the system especially in log in as well as password/username creation. The identity of user’s needs to be protected.

In order to provide reliable gameplay, it is of the utmost importance that the game state integrity be preserved during transmission especially. If the game state is to be altered, then the integrity of the entire match is compromised. Checksums for GSP to ensure that the game state has not been altered in transmission. Similarly, a checksum is used on the game score to provide a similar level of reliability.

A small, but not insignificant, way a level of security is implemented in GSP is using usernames. When a client is requesting to start a game, it does so by using the username or the alias created by the opposing client. This keeps the hostname of the opponent secure on the GSP server and thus no client is aware of anything but the alias of another client. This provides a veil of anonymity while still being secure through the use of authentication.

A recognized shortcoming of GSP is through the use of a single server to manage GSP traffic and data. All client information including, usernames and passwords, standings and game scores exist in a single location creating a target for malicious actors. Should a malicious actor gain access to the contents of the GSP server, they could potentially manipulate the data or steal it, in turn compromising the entire protocol. A logical step to mitigate this issue is to encrypt the server but in the future, it would be extremely beneficial to implement more robust methods of security. This could be done by making the server into multiple servers thereby creating a distributed network or just by dispersing information across multiple servers.

The Game State Protocol will enable two player, and eventually multiplayer gameplay in a secure and reliable fashion. In its design, extensibility and security were taken into consideration and thusly tradeoffs were made to provide such features. While it may seem like a trivial protocol when compared to something like FTP or TCP, the hope is to make it extensible and mostly agnostic so it can enable the future of technology and gaming in ways we cannot yet imagine.

Updates

Several updates have been made since the original release of this document due to issues that came to light during implementation. The first major update was to the DFA. The original DFA only had four states that were very vague in nature. During implementation it became evident that to manage and secure progression through the protocol additional states would need to be added. Additionally, there were some changes to command message formats, arguments were added in certain commands to allow more explicit messaging throughout. Lastly, do to security restrictions of using port 40 the port was changed to 1058.