Research On GameSpy Protocol

Arves100, xiaojiuwo

First Edition

January 28, 2020

Contents

Chapter 1

Introduction

1.1 The History of GameSpy

1.2 Related Works

Chapter 2

General Information

In this chapter we describe the structure of GameSpy SDK and GameSpy servers.

2.1 SDK Module

GameSpy SDK contains of 16 modules.

- Brigades
- Chat
- Presence & Messaging
- CDKey
- Stats & Tracking
- Persistent Storage
- Transport
- NAT Negotation
- Peer to Peer communication
- Patching & Tracking
- Server Browser
- Query & Reporting
- SAKE Persistent Storage
- ATLAS Competition
- Voice Chat
- Web Authentication

2.2 GameSpy Back-end Servers

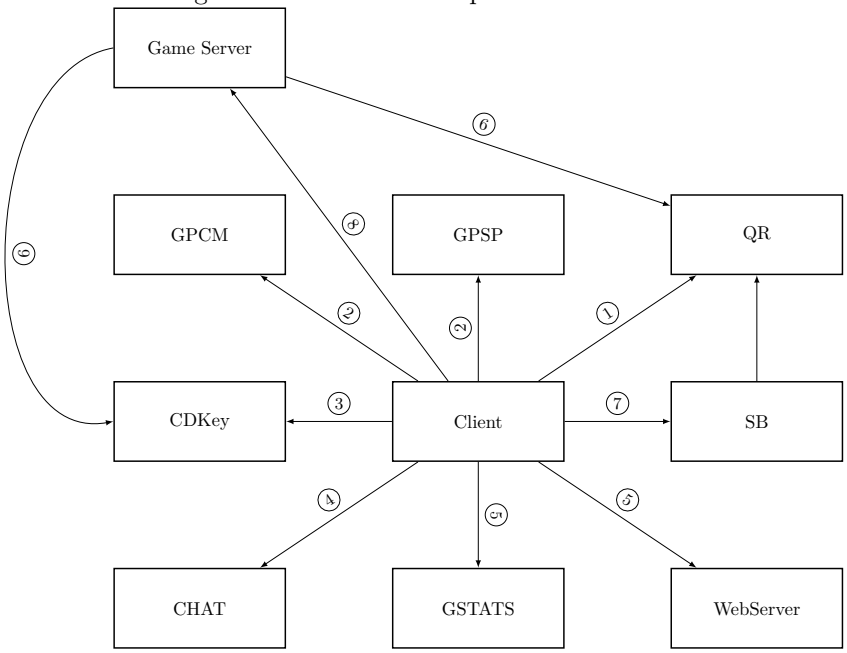
GameSpy back-end servers are list as follows.

- GameSpy Presence Connection Manager (GPCM)
- GameSpy Presence Search Player(GPSP)
- GameSpy Query and Report (QR)
- GameSpy Server Browser (SB)
- GameSpy Stats & Tracking (GStats)
- GameSpy Chat
- GameSpy NAT Negotation (NatNeg)
- GameSpy CDKey
- GameSpy Web Services
- GameSpy SAKE Storage (SAKE)

2.3 The Access Sequence of Client

If a user want to use GameSpy service, the access sequence is listed in Figure 2.1 and we describe the detail below.

Figure 2.1: The access sequence of client



Access sequence explain

1. Client access to available check in QR server, which tells client GameSpy back-end server status.
2. Client access GPCM or GPSP to check their account and login.
3. Client access to CDKey to verify his cd-key in login phase.
4. Client login to Chat server.
5. Client retrieve player data(level, exp, etc.) from GStats(old game use this server to store player data, new game use Web Server to store player data).
6. When a game server is launched it will send heartbeat to QR server to tell QR its information.
7. Client access to SB to search online game server.
8. Client login to game server with his information and cd-key.
9. Game server will check his cd-key by accessing to CDKey server, after every information is verified, client should be able to play their game.

2.4 Basic Description of Protocol

In this part, we describe some of the basic patterns that are used in all GameSpy servers.

2.4.1 The String Pattern

We first introduce the pattern of the string, which is using to make up a request and response. The following servers do use the pattern: Presence Connection Manager, Presence Search Player, GameSpy Status and Tracking, CD-Key, Query Report(version 1) This kind of string represents a value in a request and response sent by the client or the server as Table 2.1.

String	Description
$\backslash key\backslash\langle value\rangle\backslash\backslash$	The key is *key*, the value of the key is *value*

Table 2.1: String pattern

There are two kind of patterns the first one is value string, the second one is command string.

Value String This kind of string represents a key value pair in the request or response string, it has a key and a correspond value as shown in Table 2.2.

String	Description
$\backslash pid\backslash 13\backslash$	The key is *pid*, the value of the *pid* is 13
$\backslash userid\backslash 0\backslash$	The key is *userid*, the value of the *userid* is 0

Table 2.2: Value string

Command String This kind of string represents a command in a request sends by the client or the server as Table 2.3. The command will end with \\ or \ depends on whether run at the server-side or client-side.

String	Description
command\\	This is a command

Table 2.3: Command string

Chapter 3

GameSpy Presence & Messaging

Presence & Messaging system allows a game to add account authentication or registration, which includes a profile where personal information could be stored (such as email, first name), a friend list (called buddies), private messages.

GameSpy Presence contains two server, GameSpy Presence Connection Manager (GPCM) and GameSpy Presence Search Player (GPSP). GPCM is a server that manages the profiles (such as login, storing the profile information).

3.1 Common Information

In this section we describe the common information, methods, techniques that GPCM and GPSP have.

3.1.1 Server IP and Ports

Table 3.1 are the IP and Ports of GPCM and GPSP that client or game connect to.

Name	IP	Port
GPCM	gpcm.gamespy.com	29900
GPSP	gpsp.gamespy.com	29901

Table 3.1: IP and Ports for GameSpy Presence Servers

3.2 GameSpy Presence Connection Manager

3.2.1 Request For GameSpy Presence Connection Manager

Table 3.2 lists the request (known by us) that clients send to GameSpy Presence Connection Manager server (GPCM).

Commands	Description
$\backslash inviteto\backslash$	Invite friends
$\backslash login\backslash$	Login to GPCM
$\backslash getprofile\backslash$	Get the profile of a player (including your own)
$\backslash addbuddy\backslash$	Add a player to my friend list
$\backslash delbuddy\backslash$	Delete a player from my friend list
$\backslash updateui\backslash$	Update login information (email, password)
$\backslash updatepro\backslash$	Update my profile such as first name, last name, gender etc.
$\backslash logout\backslash$	Logout manually by user
$\backslash status\backslash$	Update the status of a user (Such as what game is the player playing)
$\backslash ka\backslash$	Keep client alive (do not disconnect)
$\backslash bm\backslash$	Message command

Table 3.2: Request For GameSpy Presence Connection Manager

Error response string for (GPCM, GPSP):

$$\backslash error\backslash\backslash err\backslash\langle errorcode\rangle\backslash fatal\backslash\backslash errmsg\backslash\langle errormessage\rangle\backslash id\backslash 1\backslash final\backslash \quad (3.1)$$

3.2.2 Login Command $\backslash login\backslash$

We show the login communication diagram in Fig 3.1

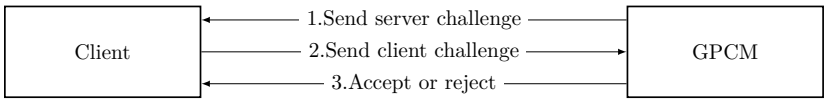


Figure 3.1: Login diagram

1. Server initial Challenge
When a client is connected to GPCM server, GPCM Server will send a challenge to client. The challenge string shows in 3.2 and 3.3. However we do not know the correct functionality of 3.3.

$$\backslash lc\backslash 1\backslash challenge\backslash\langle\rangle\backslash final\backslash \quad (3.2)$$

$$\backslash lc\backslash 1\backslash challenge\backslash\langle\rangle\backslash nur\backslash\backslash userid\backslash\langle\rangle\backslash profileid\backslash\langle\rangle\backslash final\backslash \quad (3.3)$$

- challenge: The challenge string sent by GPCM.

Keys	Description	Type
challenge	The challenge string sended by GameSpy Presence server	String
nur	? Create new user delimiter	
userid	The userID of the profile	Uint
profileid	The profileID	Uint

Table 3.3: The first type login response

2. Client Login Challenge
There are three ways of login:

- AuthToken: Logging using an alphanumeric string that represents an user.
- UniqueNick: Logging using a nickname that is unique from all the players.
- User: Logging with nickname, email and password.

We show the common part of login request in 3.4

$$\begin{aligned}&\backslash login\backslash\backslash challenge\backslash\langle\rangle\backslash \star \backslash userid\backslash\langle\rangle\\&\backslash profileid\backslash\langle\rangle\backslash partnerid\backslash\langle\rangle\backslash response\backslash\langle\rangle\\&\backslash firewall\backslash 1\backslash port\backslash\langle\rangle\backslash productid\backslash\langle\rangle\\&\backslash gamename\backslash\langle\rangle\backslash namespaceid\backslash\langle\rangle\\&\backslash sdkrevision\backslash\langle\rangle\backslash quiet\backslash\langle\rangle\backslash id\backslash\backslash final\backslash\end{aligned} \tag{3.4}$$

Where the value of $\star$ in 3.4 depending on which login method user is using.

$$\begin{aligned}&\backslash authtoken\backslash\langle\rangle\backslash\\&\backslash uniquenick\backslash\langle\rangle\backslash\\&\backslash user\backslash\langle\rangle\backslash\end{aligned} \tag{3.5}$$

Keys	Description	Type
login	The login command which use to identify the login request of client	
challenge	The user challenge used to verify the authenticity of the client	See A
authtoken	The token used to login (represent of an user)	String
uniquenick	The unique nickname used to login	String
user	The users account (format is NICKNAME@EMAIL)	String
userid	User id	Uint
profileid	Profile id	Uint
partnerid	This ID is used to identify a backend service logged with gamespy.(Nintendo WIFI Connection will identify his partner as 11, which means that for gamespy, you are logging from a third party connection)	Uint
response	The client challenge used to verify the authenticity of the client	String
firewall	If this option is set to 1, then you are connecting under a firewall/limited connection	Uint
port	The peer port (used for p2p stuff)	Uint
productid	An ID that identify the game you're using	Uint
gamename	A string that rapresents the game that you're using, used also for several activities like peerchat server identification	string
namespaceid	Distinguish same nickname player	Uint
sdkrevision	The version of the SDK you're using	Uint
quiet	? Maybe indicate invisible login which can not been seen at friends list	Uint
lt	The login ticket used for login into SAKE	String
id	The operation number	Uint

Table 3.4: Login parameter string

3. Server Response

When received client's login request, server check the challenge and proof. if client pass the check, server will first send response3.2.2 and then it will send friend list friend status, message, add friend request.

$$\backslash lc\backslash 2\backslash sesskey\backslash\langle\rangle\backslash userid\backslash\langle\rangle\backslash profileid\backslash\langle\rangle \backslash uniquenick\backslash\langle\rangle\backslash lt\backslash\langle\rangle\backslash proof\backslash\langle\rangle\backslash final\backslash \tag{3.6}$$

Keys	Description	Type
sesskey	The session key, which is a integer rapresentating the client connection	
userid	The userID of the profile	
profileid	The profileID	
uniquenick	The logged in unique nick	
lt	The login ticket, unknown usage	
proof	The proof is something similar to the response but it vary	

Table 3.5: The second type login response

Proof in 3.5 generation: $md5(password)||48spaces$ The user could be AuthToken or the User/UniqueNick (with the extra PartnerID). server challenge that we received before. the client challenge that was generated before.

3.2.3 SDK Revision

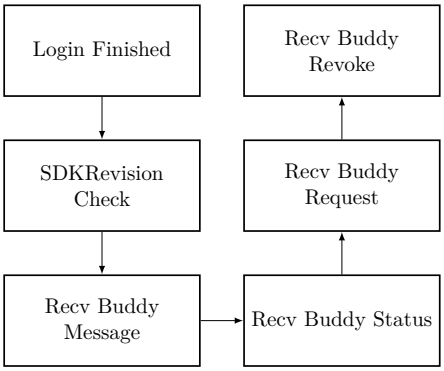


Figure 3.2: SDK Revision process

When a player finished login, GPCM will check his sdkrevision, sdkrevision is an addition of each sdkrevision number. Every addition of sdkrevision number will make GPCM act differently.

- Extended message support
  - 1 GPI_NEW_AUTH_NOTIFICATION = 1
  - 2 GPI_NEW_REVOKE_NOTIFICATION = 2
- New Status Info support
  - 4 define GPI_NEW_STATUS_NOTIFICATION = 4
- Buddy List + Block List retrieval on login
  - 8 GPI_NEW_LIST_RETRIEVAL_ON_LOGIN = 8
- Remote Auth logins now return namespaceid/partnerid on login
  - 16 GPI_REMOTEAUTH_IDS_NOTIFICATION = 16
- New CD Key registration style as opposed to using product ids
  - 32 GPI_NEW_CDKEY_REGISTRATION = 32

For now, we know the sdkrevision number of GameSpy SDK test and Crysis2.

3.2.4 Message System

3.3 GameSpy Presence Search Player

Table 3.1 are the GPSP IP and Ports that client/game connect to.

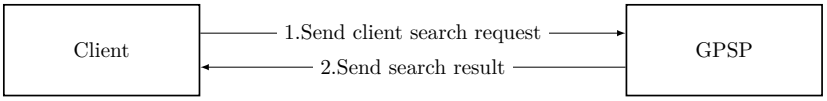


Figure 3.3: GPSP diagram

3.3.1 Search User

Client request 3.7.

$$\begin{aligned}&\backslash search\backslash\backslash sesskey\backslash\langle\rangle\backslash profileid\backslash\langle\rangle\\&\backslash namespaceid\backslash\langle\rangle\backslash partnerid\backslash\langle\rangle\\&\backslash nick\backslash\langle\rangle\backslash uniquenick\backslash\langle\rangle\\&\backslash email\backslash\langle\rangle\backslash gamename\backslash\langle\rangle\backslash final\backslash\end{aligned} \tag{3.7}$$

Server response 3.8.

$$\backslash bsr\backslash\langle profileid\rangle\backslash nick\backslash\langle nick\rangle\backslash \tag{3.8}$$

3.3.1.1 User Creation

This commmand 3.9 is used to create a user in GameSpy.

$$\begin{aligned}&\backslash newuser\backslash email\backslash\langle email\rangle\backslash nick\backslash\langle nick\rangle\\&\backslash passwordenc\backslash\langle passwordenc\rangle\backslash productid\backslash\langle productid\rangle\\&\backslash gamename\backslash\langle gamename\rangle\backslash uniquenick\backslash\langle uniquenick\rangle\\&\backslash cdkeyenc\backslash\langle cdkeyenc\rangle\backslash partnerid\backslash\langle partnerid\rangle\backslash id\backslash 1\backslash final\backslash\end{aligned} \tag{3.9}$$

The description of each parameter string is shown in Table **??**. This is the response that server sends to client:

$$\begin{aligned}&\backslash bsr\backslash < profileid > \backslash nick\backslash < nick > \backslash uniquenick\backslash < uniquenick >\\&\backslash namespaceid\backslash < namespaceid > \backslash firstname\backslash < firstname >\\&\backslash lastname\backslash < lastname > \backslash email\backslash < email >\\&\backslash bsrdone\backslash < gamespyencdeterminator > \backslash final\backslash\end{aligned} \tag{3.10}$$

Chapter 4

Persistent Storage

Chapter 5

Transport

Chapter 6

NAT Negotation

Three matchup servers: natneg1. natnet2. natneg3.

IP: natneg1.gamespy.com or natneg2.gamespy.com or natneg3.gamespy.com Protocol: UDP Port: 27901

Nat Negotiation mechanism: Because the ip address and other environment are changing from time to time, so when a client1 wants to connect to client2, he dose not know any informations about client2, so he cannot connect to client2. using natneg it can ask client2 information on gamespy nat server and connect to client2.

The Nat Negotiation do the following things: 1.connect to GameSpy nat server 2.send the data that contain all information about himself to gamespy nat server 3.GameSpy Nat server store clients information. 4.when a client1 is try to connect to other client2: (1) client1 send request to gamespy nat server (2) gamespy nat server send the information about client2 to client1 (3) client1 get the client2 information and connect.

1. Client discovers if the servers are reachable sends the following data

```
typedef struct _InitPacket
{
unsigned char porttype;
unsigned char clientindex;
unsigned char usegameport;
unsigned int localip;
unsigned short localport;
} InitPacket;

#define REPORTPACKET_SIZE BASEPACKET_SIZE + 61
typedef struct _ReportPacket
{
unsigned char porttype;
unsigned char clientindex;
unsigned char negResult;
NatType natType;
NatMappingScheme natMappingScheme;
char gamename[50];
} ReportPacket;

#define CONNECTPACKET_SIZE BASEPACKET_SIZE + 8
typedef struct _ConnectPacket
{
unsigned int remoteIP;
unsigned short remotePort;
unsigned char gotyourdata;
unsigned char finished;
} ConnectPacket;

#define BASEPACKET_SIZE 12
#define BASEPACKET_TYPE_OFFSET 7
typedef struct _NatNegPacket
{
// Base members: unsigned char magic[NATNEG_MAGIC_LEN];
unsigned char version;
unsigned char packettype;
int cookie;

union
{
InitPacket Init;
ConnectPacket Connect;
ReportPacket Report;
} Packet;

} NatNegPacket;
```

Magic: 0xFD 0xFC 0x1E 0x66 0x6A 0xB2 Version: 0x03
Command 1: Natify Request (0x12) Cookie: htonl(777)

Sets Packet Init. portType to 1 if Natneg server is natneg1, 2 if it's natneg2, 3 if it's natneg3

Command 2: Address check (discover mapping)

Sets portType to as the same as Command 1 Cookie is htonl(0 for natneg1 (map1A), 1 for natneg2, 2 for natneg3, 3 for natneg1 second map (Map1B))

Chapter 7

Peer to Peer communication

Chapter 8

Patching & Tracking

Chapter 9

Query & Reporting

Custom keys are used to define custom data to report, for example if the user is playing with a Windows or Machintosh PC.

There could be two types of custom keys: Player keys (they end with _): Custom player information Team keys (they end with _t): Custom team (or brigade) information Server keys (they don't end with anything): Custom server information

Custom keys starts from 50 to 253

IP: gamename.master.gamespy.com Port 27900 Protocol: UDP

There is more than one Query report ports, if 27900 is not found the system will try to scan the ports up to 28000

A dedicated server sends some information data to GameSpy Master Server to let GameSpy know that a new server was started, so users can find the server in the server browser like GameSpy 3D or GameSpy Arcade.

A server needs to be registred to GameSpy master Server, it's done with a challenge Sending the heartbeat challenge packet and processing the response. If an error happens, the AddError packet is sended.

A. Heartbeat (Only done if the server is public) The heartbeat checks if the dedicated server is active or not. When a dedicated servers sends a data, the time when the data is sended is saved in the Master server. If the Master server does not receive a new data in 10 seconds, Master server removes the dedicated server to the list and assumes the server is offline.

The dedicated server have to send the heartbeat packet each 10 seconds in order to maintain his connection alive.

The Instance key is a random 4 bytes array characters generated by the client when it tries to connect to the server

The heartbeat communicates everything new it happends to the server, like someone connected or similar.

Keep alive packet: A 5 bytes buffer composed by 0x08 (The packet id) Instance key

3 types of heartbeat packets Type 3: Challenge heartbeat Type 2: A server is shutting down Type 1: User requested a change in the game data Type 0: Normal heartbeat

General heartbeat packet:

0x03 (The Packet ID) Instance key

A key represents the information of a data, much like a Dictionary (Similar to

GPSP, but it uses \0 rather than \\)
List of known keys: localipX (Where X is the number of local IP starting from 0): Local IP of the server localport: Query port binded by the server, where the Master Server can connect to natneg: If you can nat negotiate with the server (If you do, the keep alive packet will also be sended) statechanged: Integer (Type of heartbeat, see above) gamename: Name of the game
If the server want to track the local clients public ip, also this two extra parameters will be sended: publicip: Public IP of the server publicport: Public port
The custom keys are now added with their respective value Server, Player and Team

NOTE: In the heartbeat, we are always querying the current known keys, so rather than being "customkey_one\0customkey_one_data\0" it's just "customkey_one\0\0" (Each key is delimited by \0)
B. Check queries (Process any new query)
We receive some data from the server.
CD-Key query: They start with 0x3B, nothing else is known See CD-KEY Reverse for more information
Query Report 1 queries (compatibility): They start with \
Nat Negotiation query: If the length is bigger than 6 and we find the NatNeg magic data See NatNeg Reverse for more information
Query Report 2: If the first two bytes are 0xFE and 0xFD
Query Report 2 Queries: Structure: Byte

$$0$$

= 0xFE Byte

$$1$$

= 0xFD Byte

$$2$$

= Packet type Byte

$$3 - 10$$

= Request key (An array long 7 bytes)
After all the queries are processed, the dedicated server sends back some data. Which can be the challenges or something different.

Packet types:

Query (0x00) This packet verify the IP of the client by checking if the random data it was sended before (With 0x09) is the same. If it isn't the server won't verify the client.

The dedicated server will send a notification to the Master Server about who authenticated and who didn't

A character from the start of the data is called EXFlags and they are used to see if the QR2 server supports different things (an example is: Split if the server supports splitting the queries)

Maximum of 7 queries can be splitted

How a query is created: A key called splitnum is created which contains the current number of key splitted The key type (server, team or player) The key data

Challenge (0x01) This packet is used for verify the server with the master server.

Calculate the challenge: First the backend option, each server can have some custom backend option, like disabling the Query Report challenge The data sended is the following:

$$2 Bytes that are the backend option with a \backslash 0$$

$$Public IP (Length of 8, readed with htonl) and Port (length of 4)$$

$$Max of 64 bytes contain a random data that will be the challenge, this is much like GPCM$$

Algorithm of calculating the challenge (Client side):

See qr2.c at line 785 (compue_challenge_response) for more information A. Encrypt the challenge with the secret key B. Encode the encrypted challenge

Echo (0x02) Simply reply the same data as the server sended

The first byte is 0x05 Then the data the server sended (max 32 bytes are allowed)

Heartbeat (0x03) Check "General heartbeat packet"

Add Error (0x04) The master server sends an error to the dedicated server For example about Server registration (Failed challenge)

Echo response (0x05) This is a response of the Echo packet that Server sended to Client Server to Client ID is 0x02, Client to Server ID is 0x05

Client Message (0x06) Sends the following data (After the packet structure)

The first byte is 0x07 (Message ACK) The other 4 bytes is the length of the message key

There can be sent a Nat negotiation packet now (With the natneg magic) Or it can be a normam data

Max 10 messages to track

0x07???

Keep alive (0x08) Ignored packet

Prequery IP Verify (0x09) [Server to Client only] Try to verify the IP of a client that connects to the server. This is only done if the user enable the IP challenge. Each new client has to verify themself with a challenge.

A new key is added to the data to send:

Chapter 10

Server Browser

Chapter 11

SAKE Persistent Storage

Chapter 12

ATLAS Competition

Chapter 13

Voice Chat

Chapter 14

Web Authentication

Chapter 15

GameSpy Status & Tracking

when game connect to GSTATS server, server will send an message to game which contains the challenge, the total length of message must bigger than 38bytes, and the challenge must bigger than 20bytes. when game received the challenge it will compute a response, the response is formed as follows. response = CRC32(<server challenge>,<length of server challenge>)||<game secret key> then game will compute the MD5 hash as MD5value = MD5(<response>,<length of response>) then encoded with Enctype3 then construct the challenge-response message as $\backslash auth\backslash\backslash gamename\backslash < gamename > \backslash response\backslash < MD5value > \backslash port\backslash < port > \backslash id\backslash < id >$

session key length (unknown) connction id = transfer ascii of sessionkey to integer

the initialization phase is finished. server challenge message length (bigger than 38-byte) server challenge length (bigger than 20-byte) $\backslash final\backslash$ is encrypted using XOR Enctype1 at the end of the challenge that sends by the server.

Appendix A

Login Proof Challenge Generation Algorithm

Appendix B

Gstats Initial Encryption

Appendix C

CDKey Server Initial Encryption