2020 MASTERSERVER REFERENCE

333networks

AN INTRODUCTION TO THE MASTERSERVER.

Reference for heartbeats, serverlists, status and implementation.

Darkelarious darkelarious@333networks.com

Contact

Document		
Title	:	An introduction to the masterserver.
Subtitle	:	Reference for heartbeats, serverlists, status and implementation.
Revision	:	0.5
Date	:	January 24, 2021
Author	:	Darkelarious
Email address	:	darkelarious@333networks.com
Status	:	draft
Contact Info	:	333NETWORKS discord website etc

Abstract

gameserver, masterserver, client, interaction, no more gamespy, 333networks implementation, extended functionality and website.

Revisions

table with date and changes. Until v1.0 ready: to-do list.

Contents

1. Introduction

For more than fifteen years, 333networks has been hosting a masterserver. A lot of design iterations were made over the years. In this document, all acquired knowledge has been collected.

1.1 Online gaming

Since the early years of classic and modern (electronic) games, people have felt a need to play together. In the eighties, text- and turn-based games found their way onto universities' mainframes and formed the base for multiplayer games. As personal computers became the norm, the introduction of the personal computer made it possible for people at home to play together on a system with split screen and four hands on a keyboard. With the invention of the internet, it became possible to play together over great distances.

1.2 Masterserver

Since 1995, a large number of game titles were released with the capability to play online together. Online games provided the need for the development of networking protocols that made it possible to play together on different systems. Some games required a direct connection with each other (peer-to-peer) whereas other games utilised a server-client hierarchy. In order to find each other on the vastness of the world wide web, technologies were developed to connect peers, game servers and clients together. One of the technologies that was developed for this purpose was a *masterserver*. This technology provides an infrastructure for game servers and game players to interact and play together.

Game developers would initially develop their own multiplayer protocol and masterserver for their game title. With the rise of *GameSpy Industries* in 1995, game developers had the possibility to choose for proprietary software and third party services to provide the multiplayer aspects, matchmaking and masterserver support for their games. With the rise of broadband internet and reliable masterserver services available, it was no longer necessary to develop their own protocols and maintain their own masterserver hardware. Many game developers followed and dependant on *GameSpy*'s services or discontinued their own masterserver infrastructure in favour of the existing services.

1.3 Loss of GameSpy

In the autumn of 2008, all masterserver services for two of the titles, *Unreal* and *Unreal Tournament 1999* published by *Epic Megagames Inc*© went down at the same time. For three days, thousands of players, were frustrated that not a single online server showed up in the server browser [1,2]. In the spring of 2010, the masterserver infrastructure became even more fragile when the rebranded *Epic Games* permanently shut down their masterserver. Effectively, thousands of gamers now depended on a single remaining GameSpy masterserver. In December 2013, *GameSpy Industries* was bought by GLU Technologies, leading to the shutdown of all *GameSpy* services on May 31, 2014 [3,4].

Game developers and publishers who had relied on *GameSpy* to provide online multiplayer support for two decades, were left empty-handed. People who bought expensive game titles were no longer able to play online in the era where connecting with each other is essential. In response to the shutdown, some game developers like *Epic Games* returned to an in-house solution. Many other multiplayer titles and their communities had no such luck.

1.4 Filling the void

In the early years of this century, a number of Unreal Tournament players and their clans worked together to display (interactive) statistics of their game servers. Through *php*-scripts it was possible to obtain server information and display this on websites for one or more individual servers. With a small group of people, we started *333networks* as a statistics website and private game server host for Unreal Tournament 1999. With more understanding of *server queries* and *network infrastructure*, 333networks started experimenting with replicating properties of the masterserver to obtain more server information for display. This lead to the development of an intrinsic masterserver that could list all cooperating Unreal Tournament servers on the website. Due to various successes, 333networks continued with the development of a fully functional masterserver with integrated website, which was completed mere months before *GameSpy* shut down in 2014. This document contains all the knowledge that was accumulated in the process of developing and expanding the 333networks version of the masterserver.

1.5 Content overview

TODO: in this section we describe what information can be found where. This is not a repetition of the table of contents, but a contextual, descriptive narration of the document itself to justify the chosen order of information. Save this for later.

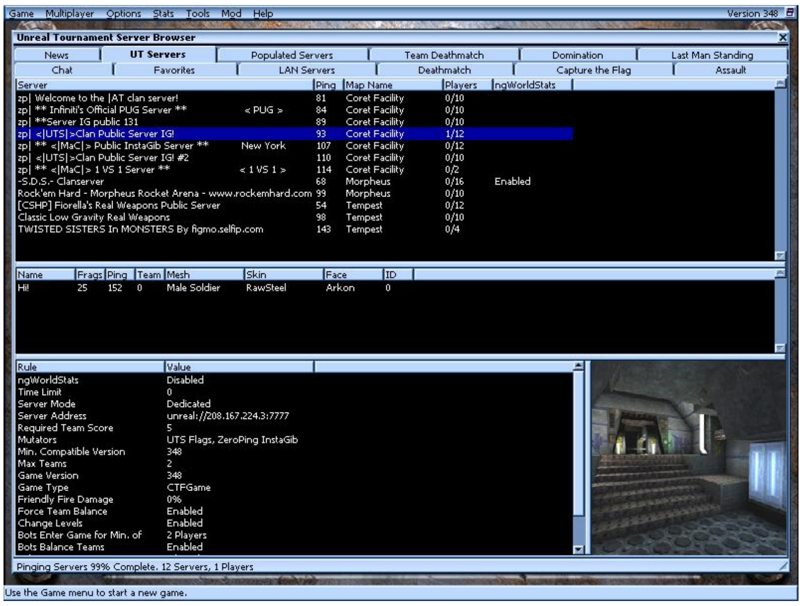
2. Background

In this chapter we introduce the concept of a masterserver and describe why legacy games depend on it.

2.1 Finding a game

The primary purpose that is to be accomplished, is to serve the paying customer, a choice in what game server he or she wants to join. Still as a business, the publisher's intended result is that a paying customer gets to play a game with other paying customers, who all involve others to buy the title and play too. The business model for this concept has changed significantly over the past two decades, but the focus is on the experience of the player.

Many games provide the option to play with or against other players. Usually there is a menu item called "Multiplayer" or "Online Game", that presents the player with a list of game servers that are currently online. A few clicks later, the player can shoot other players or conquer an evil enemy together. The focus in this document is in describing how the mechanics of acquiring this list work. To illustrate various functions and events, we use screenshots and examples from the game title "Unreal Tournament", as many of the games of that particular era are either based on or heavily influenced by this title.



Server		Ping	Map Name	Players	ngWorldStats
zp| Welcome to the |AT clan server!		81	Coret Facility	0/10	
zp| ** Infiniti's Official PUG Server **	< PUG >	84	Coret Facility	0/10	
zp| **Server IG public 131		89	Coret Facility	0/10	
zp| <|UTS|>Clan Public Server IG!		93	Coret Facility	1/12	
zp| ** <|MaC|> Public InstaGib Server **	New York	107	Coret Facility	0/12	
zp| <|UTS|>Clan Public Server IG! #2		110	Coret Facility	0/10	
zp| ** <|MaC|> 1 VS 1 Server **	< 1 VS 1 >	114	Coret Facility	0/2	
-S.D.S.- Clanserver		68	Morpheus	0/16	Enabled
Rock'em Hard - Morpheus Rocket Arena - www.rockemhard.com		99	Morpheus	0/10	
[CSHP] Fiorella's Real Weapons Public Server		54	Tempest	0/12	
Classic Low Gravity Real Weapons		98	Tempest	0/10	
TWISTED SISTERS In MONSTERS By figmo.selfip.com		143	Tempest	0/4	

Name	Frags	Ping	Team	Mesh	Skin	Face	ID
Hi!	25	152	0	Male Soldier	RawSteel	Arkon	0

Rule	Value
ngWorldStats	Disabled
Time Limit	0
Server Mode	Dedicated
Server Address	unreal://208.167.224.3:7777
Required Team Score	5
Mutators	UTS Flags, ZeroPing InstaGib
Min. Compatible Version	348
Max Teams	2
Game Version	348
Game Type	CTFGame
Friendly Fire Damage	0%
Force Team Balance	Enabled
Change Levels	Enabled
Bots Enter Game for Min. of	2 Players
Bots Balance Teams	Enabled

Pinging Servers 99% Complete. 12 Servers, 1 Players

Use the Game menu to start a new game.

Figure 2.1: The Unreal Tournament server browser with several online gameservers.

When the player opens the multiplayer functionality, a list of online servers opens. We refer to this as the *server browser*. Its main purpose is to provide the player with the available game servers and their individual properties, such as number of active players and current level or map. In figure 2.1 we see an example of such a server browser where we can identify three significant areas with information.

In the area on top, we see a list of online game servers with the name of the server, network response time (ping), name of the map and number of players. With this information, the player can decide to join their favourite server by name or just join any of the populated servers to play. The second segment displays a list of players and detailed information per player, like his/her score, ping, model (body shape) and textures (looks). With this list, the player can assess whether their friends are already playing and waiting for them, browse different servers and find anyone with whom they have played before or decide to join new people with the same interest in the game.

The last segment displays detailed server information, like the game type and playing style, team settings, server administrator details and a prominent server level screenshot. This segment provides a lot of detailed information that may be relevant to the more experienced players or people who are looking for a specific combination of mutators (game modifications). The combination of the information presented in these three segments drive the player to select a server of their choosing and join it.

2.2 Online games

The concept of online gaming and communication of this information over the internet may be a bit vague at this point. Everybody plays from their own computer and it is not always clear what interaction occurs at which side of the internet connection. In order to illustrate the different kinds of networking, we compare online games with their analogue variants: chess and poker. With chess, the two players could be playing on the same board or could be a large distance apart and communicate their moves by telephone: player one states he moves his rook from coordinates A3 to E3 and player two could respond by telling to move his pawn from C7 to C6. Both players are aware of the moves on the board and are playing the same game.

With poker, it will be a lot more difficult to keep players appraised of the same game information without telling each other which cards they are holding. This requires of an independent party, the dealer. This dealer would communicate with each player individually which cards (s)he received from the deck and whether this person raises the pot. These analogue representations of multiplayer games can be described as peer-to-peer (chess) and server-client (poker) approaches. With a peer-to-peer game, players are directly connected to each other and are both aware of all aspects in the game. With a server-client game, each player has their unique response that the others are unaware of and only the server is capable of exchanging information between the others.

From the poker analogy, we continue to multiple games. In the local bar, several groups of people are playing different rounds of poker on each table in different room. Every room has its own table, dealer and participants, independent from all other rooms and tables. In order to participate in a round of poker, you therefore only need to remember the address of the bar in order to gain access to all of the poker tables. Once you enter the bar, you ask the barkeep in which rooms the poker tables are located. The barkeep will tell you which rooms there are and you can look around the rooms to decide which table you want to join. The barkeep represents the masterserver, who keeps a list of all servers (rooms or poker tables) that are available. You do not need to maintain a record of the ever changing room/table list, instead one can just access the information directly from the barkeep.

2.3 Game interactions

To illustrate the more technical interactions between dealers, barkeep and player, we create three roles: **gameserver**, **masterserver**[1] and **client** in figure 2.2. Everybody with an internet connection and a copy of the game could host their own gameserver. When a gameserver is initialised, it repeatedly sends a signal to the masterserver (1). We refer to these signals as *heartbeats*, as the masterserver listens for these signals to determine if the gameserver is active, similar to how we listen for heartbeats in the human body to determine if somebody is still alive.

[1]We refer to *gameservers* and *masterservers* instead of *game-* and *master* servers. The latter is correct English, but the former improves readability and avoids confusion about what type of server we explicitly try to describe.

Game clients make a request to the masterserver to obtain a list of all servers that sent heartbeats and the masterserver provides this list (2). The client then inquires at all of these gameservers for their information (or *status*) so that the user can make a choice which gameserver to join (3).

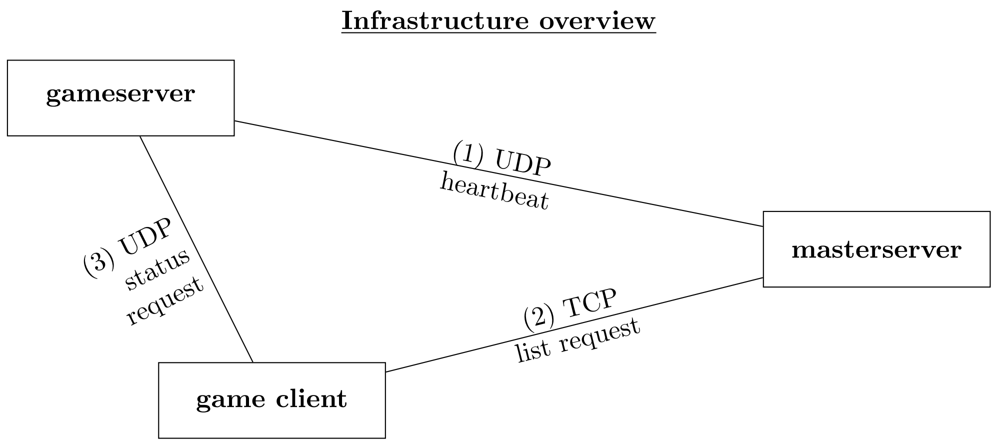


Figure 2.2: General interactions between gameserver, masterserver and client.

All interactions between gameserver, masterserver and client occur over the internet. To avoid a wild growth of different methods to convey the information, GameSpy created a format of communication to handle many games with the same approach. When different gameservers follow the same protocol, the same interaction can be applied for a large variety in games. In other words, one barkeep could keep a room list for poker, monopoly, bridge and other games, instead of having one barkeep for every game: the masterserver can handle different games that use the same form of communication.

2.4 Networking

In the next chapters we address every interaction between gameserver, masterserver and client, but first we describe what we consider to be some required general knowledge about network communication and protocol. Since there is a variety of literature available for in-depth knowledge, we limit ourselves to the essential information that is necessary to understand the following chapters.

There are multiple methods to send information over the internet between peers. The two most important methods are *UDP* and *TCP* communication. Imagine that there is a speaker in front of a large audience. In order to get to know the audience, the speaker could ask everyone to shout their names. Everyone would shout their names one by one to the speaker and hope that the speaker would hear them properly. The *user datagram protocol* (UDP) is similar to this situation: a gameserver sends a heartbeat to the masterserver, but there is no guarantee that information is correctly received or in the correct order.

The other method would be for the speaker to ask the audience to walk up to him one by one, shake hands and introduce themselves before sitting down again. The *Transfer Control Protocol* handles information in the same way: a client connects with the masterserver, identifies itself and retrieves the serverlist. As one could imagine, the information is conveyed correctly, but at the expense of a much larger duration of the interaction. The choice for either of these protocols in the game interactions is defined by the circumstances: some interactions require speed rather than precision, some interactions require precision over speed.

2.5 Protocols

Any form of communication, whether it is on the street, in the shop or digitally between two or more computers, follows a certain communication structure. During any meeting we often shake hands [2] and introduce ourselves before getting to business. This is called a *protocol*. A protocol is a set of agreements that all communicating parties follow in order to convey information.

Many of the interactions that we discuss in this document follow the *GameSpy* protocol. This protocol was, as the name suggests, developed by GameSpy Industries [5] and contains a series of steps to perform the interactions as described in figure 2.2. Many of the games published between 1995 and 2005 follow the GameSpy protocol and communicate in the same manner.

The GameSpy protocol formats the messages or data packets through backslashes. The general definition and examples of these messages are seen in example 2.1, where the first line specifies the general format. The first word is a keyword, followed by a value, both prefixed by a backslash (line 1). Multiple keys and values can be appended to contain multiple game properties in a single message (line 2).

Listing 2.1: Example of the GameSpy protocol message formatting.

```
1  \key\value
2  \key_1\value_1\key_2\value_2
3  \queryid\11.1
4  \tournament\False
5  \echo\some text
```

Digital data knows multiple data types, such as integers and floats for numbers, booleans for yes/no situations and strings for text. The GameSpy protocol has the versatility to transport all of these datatypes as plain text. Numbers can be conveyed as they are, either as integer (whole number) or float (decimal number) as seen in line 3. Boolean logic, like yes/no situations or 1/0 parameters are expressed through whole-word “True” or “False” values in line 4. Text is represented as-is and can contain spaces and special characters, as long as it is an ascii-character and not a backslash (line 5).

The GameSpy protocol specifies a number of standardised key/value pairs that address the important functions in this infrastructure as we will be discussing in the next chapters. Over the years, there have been iterations and expansions of this protocol, as well as completely new and different protocols by different third parties. The 333networks masterserver that we describe in the following chapters uses the GameSpy protocol as foundation and expands from there where necessary.

[2] This was before the global nCovid-19 pandemic.

3. Heartbeat

In this chapter, we discuss what information a heartbeat contains and how it is processed by the masterserver.

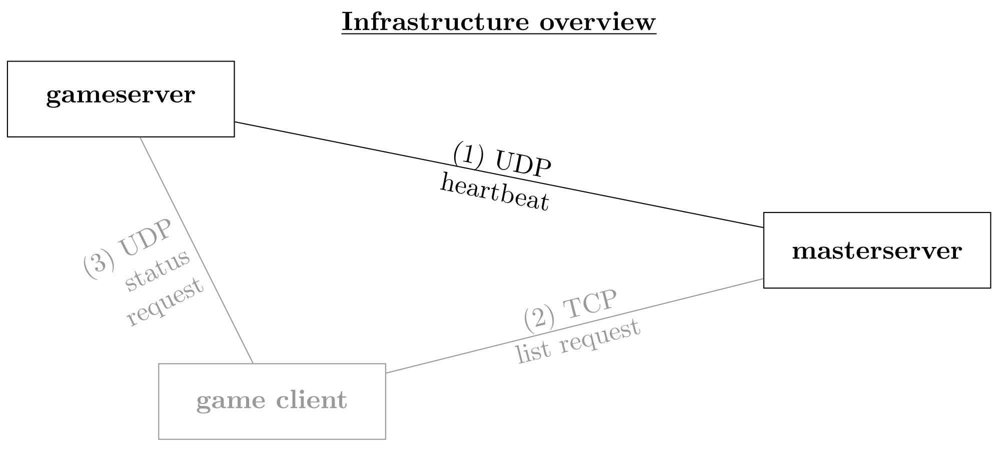


Figure 3.1: Focus on interaction between gameserver and masterserver

3.1 Exchange

When a gameserver is first initialised, it sends a signal to the masterserver as seen in figure 3.1. This message contains the gameservers IP-address, the port on which the hosted game is active and an acronym or shorthand name for the game title.

The masterserver receives this message or *heartbeat* and processes this information. As any user or automated software can send these messages, the masterserver will attempt to authenticate the game title by challenging the gameserver with a code phrase. Both the gameserver and masterserver have the necessary algorithm to determine the expected response. The gameserver returns this response and is either rejected as an invalid sender, or acknowledged as a legitimate gameserver and added to the masterserver's list of gameservers.

Heartbeats are sent repeatedly to indicate that a server is still active. The name "heartbeat" is especially similar to how we listen for heartbeats in the human body to determine if somebody is still alive. When the gameserver stops sending heartbeats, the masterserver no longer considers the gameserver active and removes it from the list.

This exchange is visualised in a diagram in figure 3.2 where the vertical axis represents passing time and the horizontal interactions correspond to the exchange discussed above.

3.2 Secure challenge

To authenticate the validity of gameserver, masterserver and client, the GameSpy protocol introduced the *secure*-challenge. In the case of a heartbeat, the masterserver attempts to authenticate the gameserver by sending it a 6-character *secure* word. Both the masterserver and gameserver

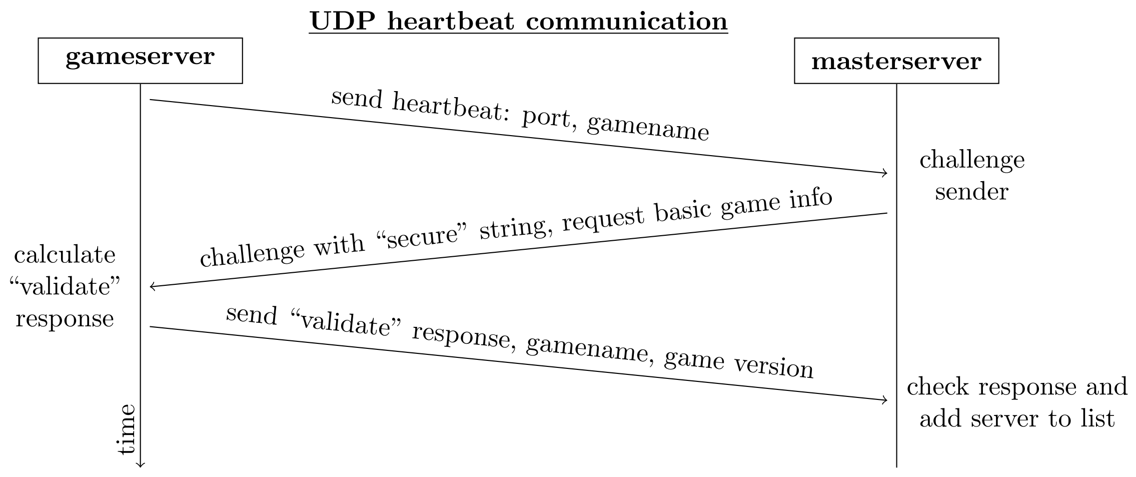

UDP heartbeat communication
gameserver
masterserver
send heartbeat: port, gamename
challenge sender
challenge with "secure" string, request basic game info
calculate "validate" response
send "validate" response, gamename, game version
check response and add server to list
time

Figure 3.2: UDP interaction between the gameserver (left) and masterserver (right) over time.

have a 6-character *cipher* that is associated with the unique game title and perform a series of mathematical operations on the combination of the secure word and cipher. The result is a unique 8-character *validate* word. If the masterserver and gameserver both arrive at the same word, the gameserver validates as legitimate and is added to the list of online gameservers.

Both the secure word and cipher may consist of any uppercase or lowercase character and any number. The validate word is a calculated response and can contain any uppercase or lowercase character and any number, but also certain special characters such as (forward) slashes, underscores and dashes. Later versions of the algorithm also allow to specify an encryption type, which results in a different validate word. The algorithm to determine the validate word consists of a series of array operations and is purely mathematical[6].

3.3 Formatting of heartbeats

Many of the games published between 1995 and 2005 follow the GameSpy protocol[5] and communicate heartbeats over the user datagram protocol (UDP). Sending data over UDP is fast at the risk of losing packets, corrupted packets or the information arriving out of order. Heartbeats contain only little quantities of information and are sent every few minutes, which minimises the chance and effect that data does not arrive properly.

Listing 3.1: Example of the GameSpy protocol message formatting.

```
1 \heartbeat\7778\gamename\ut
2 \secure\wookie\enctype\0
3 \validate\2/TYFMRc
```

A simple heartbeat is represented on line 1 in example 3.1: the `heartbeat` key indicates that the next value is the network port through which the gameserver can be queried and the `gamename` key indicates that the following value is an acronym or shorthand for the game title. An example of how the secure challenge and response would appear is seen at lines 2 and 3.

The minimum requirement for a heartbeat is an empty UDP packet. This conveys the IP-address of the gameserver. A masterserver specifically developed to support this single game title can automatically generate the default network port and game title. However, this means that the gameserver can not be serviced on another network port and that the masterserver is limited to serving a client list *only* for this specific game title.

In order to support different network ports, a heartbeat must contain a network port (as seen in example 3.1, line 2). A masterserver that supports multiple game titles also requires an additional key `gamename`. The gamename is, as mentioned before, a unique identifier for the game title and allows the masterserver to identify which game is being serviced and authenticated[3].

Additional information is often incorporated in the heartbeat as well. Some developers allow the indication of a new session through the `statechanged` keyword, that could be used to indicate to the masterserver that a new session or map has been started. An overview of the protocol is added in appendix C.

3.4 Practical execution

In the past fifteen years, we have seen many different ways of formatting a heartbeat. It is worth noting that many game developers deviate from the specified protocol. Some games like *Vietcong*, published by *Gathering of Developers* in 2003, do not specify a gamename in their heartbeat, which does not make it possible to support the game on a masterserver for multiple gamenames.

Another game, *Jetfighter IV: Fortress America* published by *TalonSoft* in 2000, was originally developed with a single masterserver in mind and thus did not follow protocol. The gameserver specifies the gamename `jetfighter4` whereas the client specifies the gamename `Jet Fighter IV` for the same game.

These inconsistencies and/or deviations from the original protocol introduce unexpected obstacles in the development process of a replacement masterserver.

[3]The game title and *gamename* both refer to the original publications from game developers. As per the protocol, we use the word “gamename” to refer to these titles from now on. This term returns in chapter 4 and 5.

4. Serverlist

In this chapter, we discuss the interaction between the game client and masterserver to retrieve a list of gameservers.

Infrastructure overview

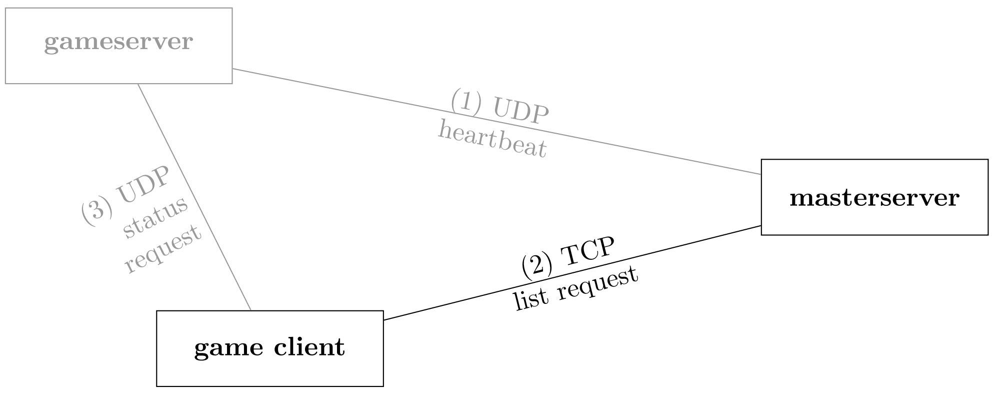


Figure 4.1: Focus on interaction between game client and masterserver.

4.1 List transfer

In the previous chapter, we discussed how heartbeats sent to the masterserver result in a complete list of online gameservers at the side of the masterserver. In this chapter we discuss how the list of gameserver addresses is transferred to the game client as seen in step (2) in figure 4.1. The request is initiated when the user opens the *server browser* in the game environment (further discussed in chapter 5). The serverlist merely contains a list of IP-addresses and query ports. When the client requests the serverlist from the masterserver, the list of addresses is transmitted.

In contrary to heartbeats, which are sent over UDP, the serverlist is sent over TCP. The integrity of the rather large serverlist is important and must be preserved during the transfer, usually split over multiple packets. To protect the data, GameSpy preferred the slower but more reliable TCP transfer for this part in the process.

4.2 Procedure

As soon as the user requests the serverlist, the client initiates a request to the masterserver by opening a connection to one or more of the configured masterservers. In listing 4.1, the exchange of the GameSpy protocol is detailed. After a TCP connection is established, the masterserver opens the exchange with a request for the basic client information (gamename, game version) and authentication via the *secure* challenge that was also used by the masterserver to authenticate heartbeats (line 1).

The client returns the basic game information and generated validate keyword in the same packet to the masterserver. As the client and the masterserver both possess the game cipher and secure keyword, as well as the gamename from the masterserver's request for basic client information, both the masterserver and the client can calculate the correct *validate* keyword to complete the authentication (lines 2 and 3).

Listing 4.1: Example of the GameSpy protocol message formatting.

```
1  \basic\\secure\wookie\final\
2  \gamename\ut\gamever\348\minnetver\348\location\0\queryid\11.1
3  \validate\2/TYFMRc\final\\queryid\11.2
4  \list\\gamename\ut
5  \list\cmp\gamename\serioussam
6  \ip\255.255.255.255:7778\ip\255.255.255.255:12556\...\final\
```

The client then proceeds with requesting the serverlist and the format in which the client would like to receive this information (line 4 or 5). If the client properly authenticated with the secure/validate keyword, the masterserver will start sending the serverlist over multiple packets. This could be a plaintext list, or compressed list as described in the following section (line 6). The masterserver appends the `final` keyword to serverlist, to indicate that the last address has been received. Immediately after sending the last packet, the masterserver closes the TCP connection.

The client will continue to receive data until the `final` keyword has arrived. If due to network disruption the `final` does not arrive, the client will attempt to receive until a timeout occurs. The addresses that are intact are processed for display to the user, missing addresses are ignored. Many games do *not* throw an error or warning when the received list was incomplete, but fail silently.

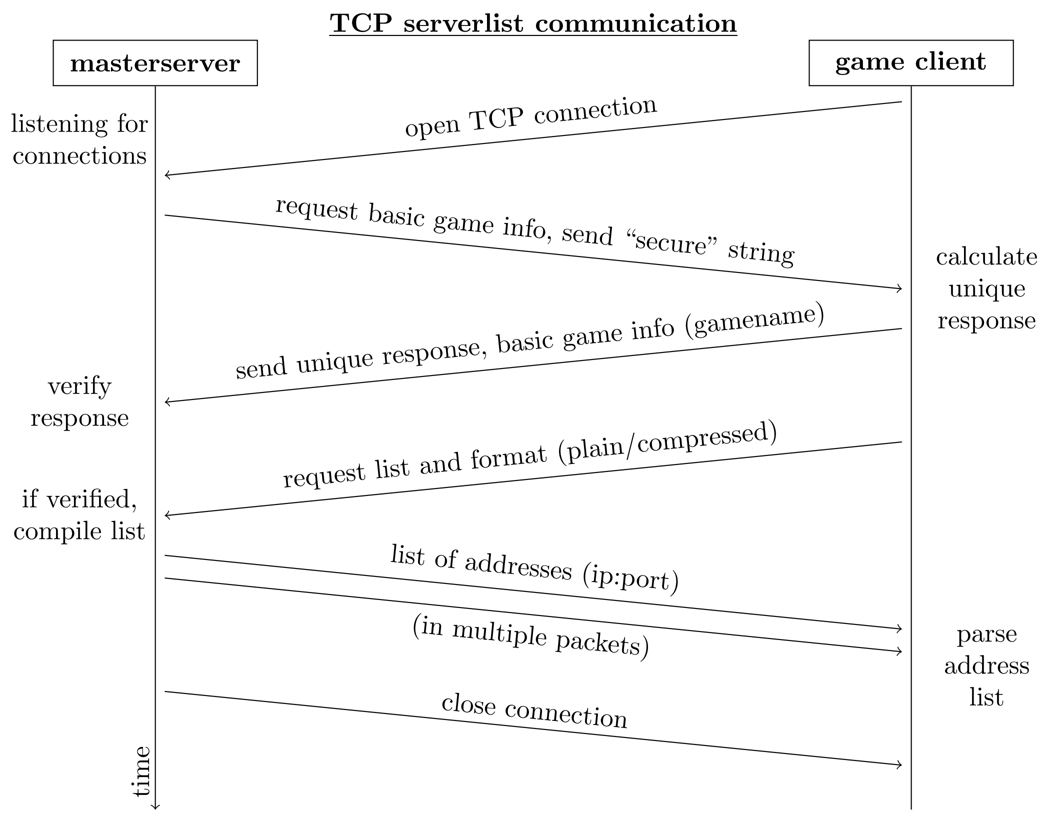

TCP serverlist communication
masterserver
game client
listening for connections
open TCP connection
request basic game info, send "secure" string
calculate unique response
send unique response, basic game info (gamename)
verify response
request list and format (plain/compressed)
if verified, compile list
list of addresses (ip:port)
(in multiple packets)
parse address list
close connection
time

Figure 4.2: TCP interaction between the masterserver (left) and game client (right) over time.

4.3 Compressed serverlist

Gameserver addresses are usually expressed as an IP-address and query port with a colon (`:`) as separator in a plaintext string. The length of this string is up to 21 characters, with a 4- or 5-digit query port as seen in the addresses at line 6 in listing 4.1. To separate multiple addresses, the `ip` keyword was added for every new address in the list, adding another 4 characters to the plaintext address. Every character requires a single byte to transfer. In the time that the GameSpy protocol was drafted, bandwidth was scarce. With more than two thousand servers during its popular period, a serverlist for Unreal Tournament could easily be $50,000$ characters large and take several seconds to transmit on a dial-up connection.

To reduce the size of the data stream, the *compressed* serverlist was provided as alternative. By specifying the `cmp` value for the `list` keyword (line 5), the masterserver would send a compressed serverlist where the gameserver addresses were not sent as plaintext, but as byte values.

A single byte can hold 2^8 or 256 values. With the inclusion of the zero, that leads to a value range between 0 and 255, including both values. An IP-address consists of four octets (groups of numbers) with the values 0 to 255, joined by periods. Conveniently, every individual octet can fit inside a single fixed-length byte, despite the lexical length being two (e.g. 99) or three (e.g. 101) digits. As we know that every new byte is a new octet, it is no longer necessary to join the octets by periods in their text representation.

\	i	p	\	2	5	5	.	2	5	5	.	2	5	5	.	2	5	5	:	1	2	5	5	6
					0xFF				0xFF				0xFF				0xFF				0x31		0x0C	

Table 4.1: Data allocation in a gameserver address (every cell represents 1 byte).

The query port consists of a number larger than 255 and can therefore not be represented with a single byte. The concept of splicing the query port number into two new numbers is difficult to grasp when we consider the decimal counting system. Therefore we make a detour through the hexadecimal counting system: a single hexadecimal digit can hold 16 values, ranging `0x0-0xF` or 0–15[4]. Two hexadecimal digits can hold the values between 0x00 and 0xFF or 0–255. With the understanding that two hexadecimal digits can be represented exactly in one single byte, we can now express any number larger than 255 in pairs of hexadecimal digits and thus in multiple bytes.

The query port is an *unsigned short integer* and requires 2 bytes to be represented. The query ports 7778 and 12556 can now be represented as `0x1E62` and `0x310C`, where the first byte holds the value `0x1E` and the second byte `0x62`. In table 4.1 we represented the same IP-address as plaintext (top) and hexadecimal values (bottom), where every occupied table cell is a byte. As seen, the bottom row requires significantly less bytes to store the same data.

Earlier we derived that a plaintext gameserver address with `ip` key prefix can occupy up to 25 characters/bytes. With the hexadecimal representation, we compressed the same address to a fixed length of 6 bytes, which means that the same Unreal Tournament serverlist could now be compressed into a $12,000$-byte data string, a reduction of more than two-thirds of the original plaintext data transfer.

[4]to avoid confusion between different counting bases, we use the prefix `0x` for hexadecimal numbers. This prefix holds no numerical value. The first two digit in the value `0x1E62` is `1E` and not `0x`.

5. Server status

client-gameserver interaction. status, info, rules, relevant information, what purpose does it serve, etc[7].

Infrastructure overview

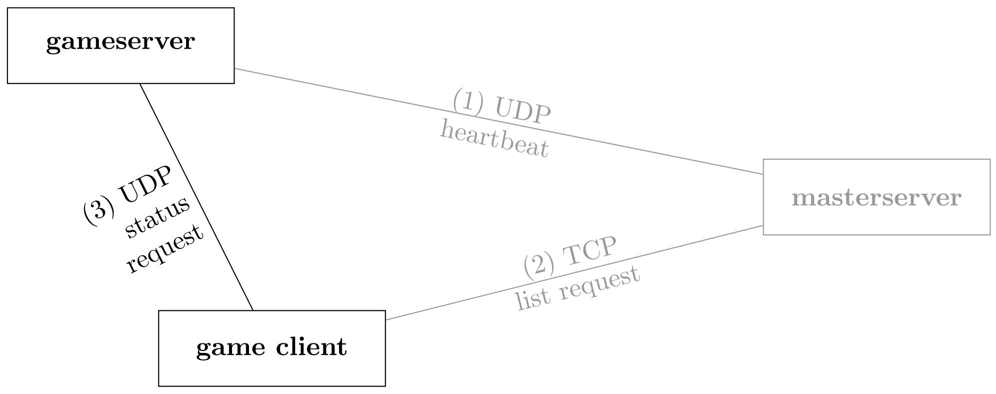


Figure 5.1: Focus on interaction between client and gameserver

focus on transferring the detailed information from the gameserver to the game client and populating the fields in the server browser interface.

UDP status communication

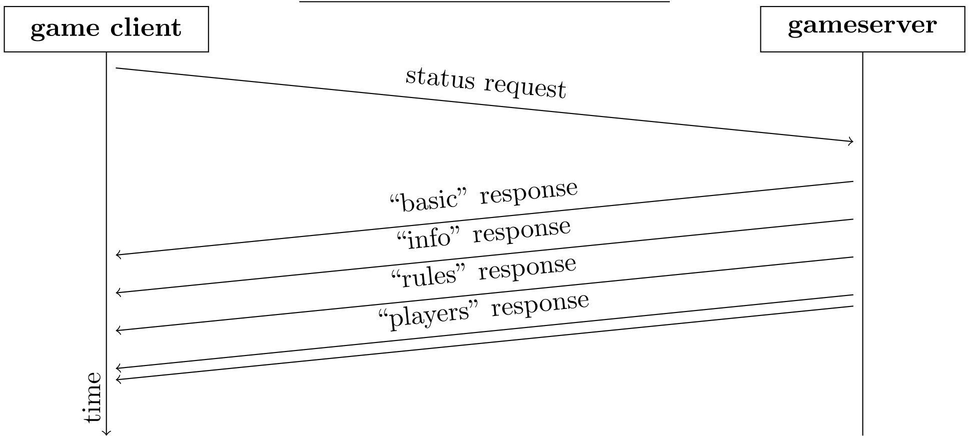


Figure 5.2: UDP interaction between the game client (left) and gameserver (right) over time.

discussion of common fields and the information they represent, player information and how using backslashes in hostnames/playernames breaks part of the query.

6. Implementation

7. Website

Addition to the concept of the masterserver: website. Browse before joining, browse without starting the game and querying all servers individually, providing a fast web reference.

Provides additional interfaces: JSON api, ip lookup, verification, method of being added. some items implemented as masterserver function, but created for the website.

Bibliography

[1] Leo(T.C.K.). Lamespy master server down. https://forums.beyondunreal.com/threads/lamespy-master-server-down.180333/, September 2008.

[2] Mr.Minus. Master Server Down In Eastern US. http://www.oldunreal.com/cgi-bin/yabb2/YaBB.pl?num=1221584697, September 2008.

[3] Christian Nutt. GameSpy ceasing all hosted services this May. http://www.gamasutra.com/view/news/214700/GameSpy_ceasing_all_hosted_services_this_May.php, April 2014.

[4] Dan Stapleton. Goodbye, and thank you from the gamespy team. http://pc.gamespy.com/articles/122/1227460p1.html, February 2013.

[5] GameSpy Industries. The GameSpy Open Architecture for Free Internet Gaming Developer Specification, 1999.

[6] Luigi Auriemma. GameSpy Masterserver Algorithm 0.3.3. http://aluigi.altervista.org/papers.htm#gsmsalg.

[7] Brandon "GreenMarine" Reinhart. All you ever wanted to know about the master uplink settings in unreal - ipserver package for unreal v1.0. Published on http://www.oldunreal.com/wiki/index.php?title=All_you_ever_wanted_to_know_about_the_Master_Uplink_settings_in_Unreal_-_IpServer_Package_for_Unreal_v1.0 by SweetFrog, September 1998.

Acknowledgements

Acknowledgements for everybody who contributed. Company names/people first, then people who actively contributed/edit and fact checking, helped with images/figures, then people who proofread it.

A. User Manual

Readable implementation of the README file.

B. Issues

Open (and later on closed) issues with the masterserver and how to solve them.

C. Protocol Reference

D. Game Overview

List of gamenames (type:longtable)