Warcraft III Game Packet Specs

v0.3 by Soar Qin

1. Packets to describe

a) Types of Warcraft III Packets

i. LAN UDP Packets.
All of this kind of packets are used to broadcast/check game information in preparation stage.

ii. In-Game TCP Packets.
These packets are used in preparation and playing stages.
I will split them into 2 sections for description.

iii. Battle.net TCP/UDP Packets.
I would head for a new doc on this part if have enough spare time.
Currently please check http://www.bnetdocs.org.

b) Only first 2 types of packets are discussed in this specs.

2. Packet header

All packets used in Warcraft III have 4 bytes header as below:

Bytes / Type	Usage
1 / uint8	Magic byte 0xF7 -- LAN UDP or Game TCP (all of packets in this doc are using this magic byte) 0xFF -- Battle.net
1 / uint8	OP Code. Check section 3, 4 and 5 for details.
2 / uint16	Packet length including this 4 bytes' header.

3. LAN UDP Packets

a) OP: 0x2F

The game sends this packet to query for LAN games, it can be used in 2 cases: 1. Reply to a 0x31 or 0x32 packet to query certain game information. 2. Sent when enter LAN Game UI. Broadcast to 255.255.255.255 to query any available games.

Bytes / Type	Usage
4 / uint32	Little-Endian ordered game type 'W3XP' = TFT 'WAR3' = ROC
4 / uint32	Game version. For example: In war3 1.24, this field is 24.
4 / uint32	Game ID, This field is zero when it is broadcasted.

b) OP: 0x30

Sent when received UDP 0x2F packets, this pakcet contains complete game information as reply.

Bytes / Type	Usage
4 / uint32	Little-Endian ordered game type
4 / uint32	Game ID
4 / uint32	System tick counts (GetTickCount() from windows)
N / 0-terminated string	This is an encoded string which contains a lot of important information of the game. Check additional notes.
4 / uint32	Number of slots
4 / uint32	Game flags, 0x01 = Scenario and 0x09 = Custom Game

	in my knowledge.
4 / uint32	Number of in-game players
4 / uint32	Number of non-computer slots So from above description, we can get this formula: Displayed in-game player number = in-game players + (slots - non-computer slots)
4 / uint32	Unknown, normally 0-0x80
2 / uint16	TCP game port listening to accept connections, in Little-Endian order.

Encoded string part:

Encoded string part:

Every even byte-value was incremented by 1. So all encoded bytes are odd. A control-byte stores the transformations for the next 7 bytes.

Since all NullBytes were transformed to 1, they will never occure inside the encoded string. But a NullByte marks the end of the encoded string.

The encoded string starts with a control byte.

The control byte holds a bitfield with one bit for each byte of the next 7 bytes block. Bit 1 (not Bit 0) corresponds to the following byte right after the control-byte, bit 2 to the next, and so on.

Only Bit 1-7 contribute to encoded string. Bit 0 is unused and always set.

Decoding these bytes works as follows:

If the corresponding bit is a '1' then the character is moved over directly.

If the corresponding bit is a '0' then subtract 1 from the character.

After a control-byte and the belonging 7 bytes follows a new control-byte

until you find a NULL character in the stream.

Example decompression code (in 'C'):

```
char* EncodedString;
char* DecodedString;
char  mask;
int   pos=0, dpos=0;
while (EncodedString[pos] != 0)
{
  if (pos%8 == 0) mask=EncodedString[pos];
  else
  {
    if ((mask & (0x1 << (pos%8))) == 0)
      DecodedString[dpos++] = EncodedString[pos] - 1;
    else
      DecodedString[dpos++] = EncodedString[pos];
  }
  pos++;
}
```

Alternatively one could interprete the encoding scheme as follow:

Bit 0 of every character was moved to the control byte and set to 1 afterwards.

Structure of decoded structure:

Bytes / Type	**Usage**
4 / uint32	Game Settings
1 / uint8	Always zero
2 / uint16	Map width
2 / uint16	Map height
4 / uint32	Map native checksum
N / 0-terminated string	Map name
N / 0-terminated string	Host username
1 / uint8	Always zero (Also a 0-terminated string?)

20 / uint8[20]	Map native SHA-1 hash, this field is added since 1.23 to avoid map dummy hack.

For checksum and SHA-1 hash, I will discuss algorithms in other articles.

c) OP: 0x31

This packet is broadcasted to 255.255.255.255 when a game is hosted.

Bytes / Type	**Usage**
4 / uint32	Little-Endian ordered game type
4 / uint32	Game version
4 / uint32	Game ID (starts from 1, and auto-increases on each game creation)

d) OP: 0x32

This packet is broadcasted to 255.255.255.255 when number of in-game players is changed.

Bytes / Type	**Usage**
4 / uint32	Game ID
4 / uint32	Displayed in-game players
4 / uint32	Total number of game slots

e) OP: 0x33

This packet is broadcasted to 255.255.255.255 when a game is cancelled.

Bytes / Type	**Usage**
4 / uint32	Game ID

4. Preparation TCP Packets

a) OP: 0x01

Sent to keep alive, it is also used in gameplay.

Bytes / Type	**Usage**
4 / uint32	System tick counts (GetTickCount() from windows)

b) OP: 0x04

Sent to new joining player to tell him information of all slots and his position.

Bytes / Type	**Usage**
2 / uint16	Bytes of following slot informations up to "Number of start spots", excluding this field's 2 bytes.
1 / uint8	Number of following slot records, define it as RN
9 * RN / Slot record * RN	Each slot record stands for detailed information of a slot. The record structure is described as below: uint8 player ID, 0 for computer, 0xFF for empty. uint8 map download progress, 0-100, or 0xFF as '?' uint8 slot status, 0-empty, 1-closed, 2-taken uint8 computer flag, 0-human, 1-computer uint8 team, 0-11, or 12 as ob/referee uint8 color, 0-11, or 12 as ob/referee uint8 race, 0x01-human, 0x02-orc, 0x04-nightelf 0x08-undead, 0x20-random with 0x40 means the race is not fixed by map. uint8 AI level, 0-easy, 1-normal, 2-insane uint8 handicap, normally 50, 60, 70, 80, 90 or 100. Can be other values used in Bot GHost++.
4 / uint32	Initial random seed of game.
1 / uint8	Team and Race locking flags:

	0x01 - Team is locked 0x02 - Race is locked 0x04 - Race is fixed to random (cannot use with 0x02)
1 / uint8	Number of start spots on map
1 / uint8	Slot ID allocated for new joining player
16 / sockaddr_in	The remote address of player seen from host by getpeername()

c) OP: 0x06

Bytes / Type	Usage
4 / uint32	Always 0x02, Maybe number of tail sockaddr_in structures?
1 / uint8	Player ID shown in slot info
N / 0-terminated string	Player name, no longer than 16 bytes including the terminating zero byte.
1 / uint8	Number of following extra bytes, always 1 in LAN
N / uint8[N]	Extra bytes, always a zero in LAN
16 / sockaddr_in	sockaddr_in from getpeername() to this player
16 / sockaddr_in	sockaddr_in from NAT routing if available (seems only used in Battle.Net).

d) OP: 0x07

Sent to all players to inform that a player is leaving the game. Also used in gameplay.

Bytes / Type	**Usage**
1 / uint8	Player ID
4 / uint32	Leave reason flags, need more research on this field.

e) OP: 0x08

Sent to all players to inform that the player is finished loading game(when host receives a 0x23 packet from client), thus players can see that the player's name background bar becomes green on loading screen.

Bytes / Type	**Usage**
1 / uint8	Player ID

f) OP: 0x09

This packet is almost the same as 0x04, but without the last 2 fields. It is sent to all players when any information about slots are changed (e.g. Map downloading, slots moving, etc.).

Bytes / Type	**Usage**
2 / uint16	Bytes of following slot informations up to "Number of start spots", excluding this field's 2 bytes.
1 / uint8	Number of following slot records, define it as RN
9 * RN / Slot record * RN	Each slot record stands for detailed information of a slot. The record structure is described as below: uint8 player ID, 0 for computer, 0xFF for empty. uint8 map download progress, 0-100, or 0xFF as '?' uint8 slot status, 0-empty, 1-closed, 2-taken uint8 computer flag, 0-human, 1-computer uint8 team, 0-11, or 12 as ob/referee uint8 color, 0-11, or 12 as ob/referee uint8 race, 0x01-human, 0x02-orc, 0x04-nightelf 0x08-undead, 0x20-random with 0x40 means the race is not fixed by map.

	uint8 AI level, 0-easy, 1-normal, 2-insane uint8 handicap, normally 50, 60, 70, 80, 90 or 100. Can be other values used in Bot GHost++.
4 / uint32	Initial random seed of game.
1 / uint8	Team and Race locking flags: 0x01 - Team is locked 0x02 - Race is locked 0x04 - Race is fixed to random (cannot use with 0x02)
1 / uint8	Number of start spots on map

g) OP: 0x0A

Only 4 bytes header, sent when host clicks Start Game button to count 5 sec down before entering loading screen.

h) OP: 0x0B

Only 4 bytes header, sent when 5 sec count down is over, to inform all players to shift to loading screen.

i) OP: 0x0F

Sent to chat receivers.

If this is a private/team chat, and the sender and receiver can connect through TCP, this packet is sent directly.

Otherwise the player will send a 0x28 packet to host first and host send this packet to all receivers.

Note that some platforms block all client-client connections so that all chats are going through host, which may cause security risk to leak private chats.

Bytes / Type	**Usage**
1 / uint8	Count of players to send chat to, define as PN
PN / uint8 * PN	Each uint8 stores a Player ID to send chat.
1 / uint8	Chat sender's Player ID
1 / uint8	Chat Flag, 0x10-preparation, 0x20-gameplay
4 / uint32	Send Flag. 0x00 - To All 0x01 - To Allies 0x02 - To OB/Referee 0x03-0x0E - To (PlayerID + 2) This field is emitted if chat flag is 0x10, aka when in preparation stage.
N / 0-terminated string	Chat string.

j) OP: 0x1B

Only 4 bytes head, sent to player to tell him/her that you are disconnected from game, even on actively leaving game. Also used in gameplay.

k) OP: 0x1E

This is the first packet sent to host when a player joins a game, containing the player's information.

Bytes / Type	**Usage**
4 / uint32	Game ID
4 / uint32	Tick counts since game starts

1 / uint8	Always zero
2 / uint16	Client's game port, used for map data exchange, direct private talk, host change, etc.
4 / uint32	Client session counter, starts from 1, and auto-increases on each game joining.
N / 0-terminated string	Username
2 / uint16	Available public ports. Define it as P
16 * P / sockaddr_in * P	There is a socket sockaddr_in structure for each public port.

l) OP: 0x21

Sent to host to inform an actively leave (host always replis a 0x1B to do disconnect). Also used in gameplay.

Bytes / Type	**Usage**
4 / uint32	Reason Flag, need more research.

m) OP: 0x23

Only 4 bytes header. Tells host that the player finished game loading.

n) OP: 0x28

This packet has 2 functions:

i. Send a chat action to host, host will redirect this chat to those target players which also used in gameplay.

ii. Make changes to slots in preparation stage.

Bytes / Type	**Usage**
1 / uint8	Count of players to send chat to, define as PN
PN / uint8 * PN	Each uint8 stores a Player ID to send chat.
1 / uint8	Chat sender's Player ID
1 / uint8	Chat Flag: 0x10 - Chat in preparation 0x11 - Change team 0x12 - Change color 0x13 - Change race 0x14 - Change handicap 0x20 - Chat in gameplay

For Chat flag = 0x10 or 0x20:

Bytes / Type	Usage
4 / uint32	Send Flag. 0x00 - To All 0x01 - To Allies 0x02 - To OB/Referee 0x03-0x0E - To (PlayerID + 2) This field is emitted if chat flag is 0x10, aka when in preparation stage.
N / 0-terminated string	Chat string.

For Chat flag = 0x11-0x14:

Bytes / Type	Usage
1 / uint8	Data. Chat flag = 0x11: Team,

	Chat flag = 0x12: Color, Chat flag = 0x13: Race Chat flag = 0x14: Handicap Check x04/0x09 slot info part.

o) OP: 0x3D

Sent from host to ask if joining player has certain map for playing.
Client will reply 0x42 packet.

Bytes / Type	**Usage**
4 / uint32	Map ID (?)
N / 0-terminated string	Map path
4 / uint32	Map file size in bytes
4 / uint32	Map whole file CRC32
4 / uint32	Map native checksum
20 / uint8[20]	Map native SHA-1 hash, this field is added since 1.23 to avoid map dummy hack.

p) OP: 0x3F

Sent to a player who don't have the map (when play sends a 0x42 packet to host with available map size < real map size), also sent between players if they can connect to each other. (I noticed there may be a 0x3E packet to query map from players other than host, but yet tested)

Bytes / Type	**Usage**
4 / uint32	Map ID (?)
1 / uint8	Player ID

q) OP: 0x42

Sent to host to report map available or download progress

Bytes / Type	**Usage**
4 / uint32	Map ID (?)
1 / uint8	Download from Player ID or Host ID
4 / uint32	Available map file size (download progress)

r) OP: 0x43

Sent map data to a player, aka transfer map to a player.

Bytes / Type	**Usage**
1 / uint8	To Player ID
1 / uint8	From Player ID
4 / uint32	Map ID (?)
4 / uint32	Offset of file, for data to write in
4 / uint32	CRC32 of following data
N / uint8[N]	Map data to packet end.

s) OP: 0x44

Replies to 0x43 packet to report downloaded size as acknowledge.
Note that map sender won't wait for this packet before sends a next 0x43 map transfer packet.

Bytes / Type	**Usage**
1 / uint8	To Player ID
1 / uint8	From Player ID
4 / uint32	Map ID (?)
4 / uint32	Downloaded map size

t) OP: 0x45

This packet replies 0x3F packet to refuse download map from the player.

Bytes / Type	**Usage**
1 / uint8	To Player ID
1 / uint8	From Player ID
4 / uint32	Map ID (?)

u) OP: 0x46

When receives 0x01 keep alive packet, reply this packet, it is also used in gameplay.

Bytes / Type	**Usage**
4 / uint32	The same value from 0x01 packet

5. Gameplay TCP Packets

a) OP: 0x0C

Gameplay packets, sent to all players every 100 ms in LAN (250 ms in Battle.net and there are tools to modify this value).

Bytes / Type	Usage
2 / uint16	Time elapsed, in milliseconds.
2 / uint16	Lowere 16 bits of CRC32 on all following bytes. Note: if gameplay data is empty, this field is emitted so that the packet is only 6 bytes long.
N / uint8[N]	Gameplay data to packet end.

If the gameplay data exceeds 1446 bytes, the game data will be splitted into several 0x48 packets and a 0x0C one as last packet to avoid MTU problems caused by some routers.

Gameplay data is the same as that in replay, so please check following docs for format:

http://www.repking.com.cn/w3g_actions.txt

http://www.repking.com.cn/w3g_format.txt

b) OP: 0x0E

Seldom seen this packet, found when game data are unsynchronized. Need more research.

c) OP: 0x14

Sent to a player when he/she is actively kicked by host. (e.g. Host leaves game)

Bytes / Type	**Usage**
1 / uint8	Player ID

d) OP: 0x48

Gameplay packet without time elpased, used when game data are splitted.

Bytes / Type	**Usage**
2 / uint16	Time elapsed, in milliseconds. Always zero here.
2 / uint16	Lowere 16 bits of CRC32 on all following bytes. Note: if gameplay data is empty, this field is emitted so that the packet is only 6 bytes long.
N / uint8[N]	Gameplay data to packet end.

e) OP: 0x26

Sent to host when did some actions. Host will collect all players' actions and reply 0x0C/0x48 packets with all of these actions.

Bytes / Type	**Usage**
N / uint8[N]	Player action data, check packet 0x0C for more.

f) OP: 0x27

Synchronization packet, sent to host when receives 0x0C gameplay data packet.

Bytes / Type	Usage
4 / uint32	Synchronization hash.

Host will keep a hash locally and check each player's 0x27 packet. If insync occurs, which is that the hash value in 0x27 packet mismatches what host calculated, the player would be disconnected in no time.