Server Discovery for Quake III Arena, Wolfenstein Enemy Territory and Quake 4

G. Armitage
Centre for Advanced Internet Architectures, Technical Report 070730A
Swinburne University of Technology
Melbourne, Australia
garmitage@swin.edu.au

***Abstract*—This report briefly summarises the server-discovery protocol used by Quake III Arena, Wolfenstein Enemy Territory and Quake 4. When requested by the player, a game client issues a query to a well-known 'master server', which replies with a list of registered (and nominally active) game servers. The client then probes each game server in sequence, presenting the player with information about each game server as replies come back. Game servers participate in this process by registering with the master server when they start up. The server discovery process is similar for all three games, based on short UDP packet exchanges. A discussion of UDP traffic between game servers and clients during connection and actual game-play is beyond the scope of this report.**

***Index Terms*—Quake III Arena, Wolfenstein Enemy Territory, Quake 4**

I. Introduction

Internet based multiplayer First Person Shooter (FPS) games typically operate in a client-server mode, with game servers being hosted by internet service providers (ISPs), dedicated game hosting companies and individual enthusiasts. Although individual FPS game servers typically only host from 4 to around 30+ players, there are usually many thousands of individually operated game servers active on the Internet at any given time. This presents a challenge - how do game clients locate up-to-date information about all the servers available at any given time, such that the player can select a suitable server on which to play. This report summarises the sequence of events that makes up server discovery for Quake III Arena (Q3A) [1], Wolfenstein Enemy Territory (ET) [2] [3], and Quake 4 (Q4) [4]. (Further details of the UDP traffic between game servers and clients during subsequent connection and actual interactive game-play is beyond the scope of this report. Details of the Quake III Arena engine's network protocol may be found at http://hobbshouse.org/wiki/index.php/ [5], along with notes on the *ioquake3* variant of the GPL'd Quake III Arena source code.)

Server discovery operates similarly for many FPS games, due to the decentralised and ad-hoc nature of FPS game server hosting. First, a game client queries a master server unique to the particular game (a server whose IP address is pre-configured into the game client software). The master server returns a list of hundreds (or thousands) of IP addresses and port numbers representing game servers who've registered themselves as 'active'. The client then steps through this list, probing each listed game server for information (such as about current map type, game type and number of players - typically a brief UDP packet exchange). As a side-effect of this probe the client also estimates the RTT between itself and each game server. All this information is presented to the player (usually as it is gathered), who then selects a game server to join.

Server discovery is usually triggered explicitly by the human player running a particular game client. It may be triggered once or multiple times to refresh the list of available servers presented to the potential player by their client-side server browser. A given client will send out hundreds or thousands of probe packets to find and join only one game server.

This report is organised as follows. Section II describes the server discovery process and packet exchanges for Quake III Arena, whilst section III describes the similar server discovery process for Wolfenstein Enemy Territory. Section IV describes the closely-related server discovery process and packet exchanges for Quake 4. The report concludes in Section V.

II. Quake III Arena

Quake III Arena was released in December 1999, and was primarly focussed on multiplayer online game play. First we will review the sequence of events from

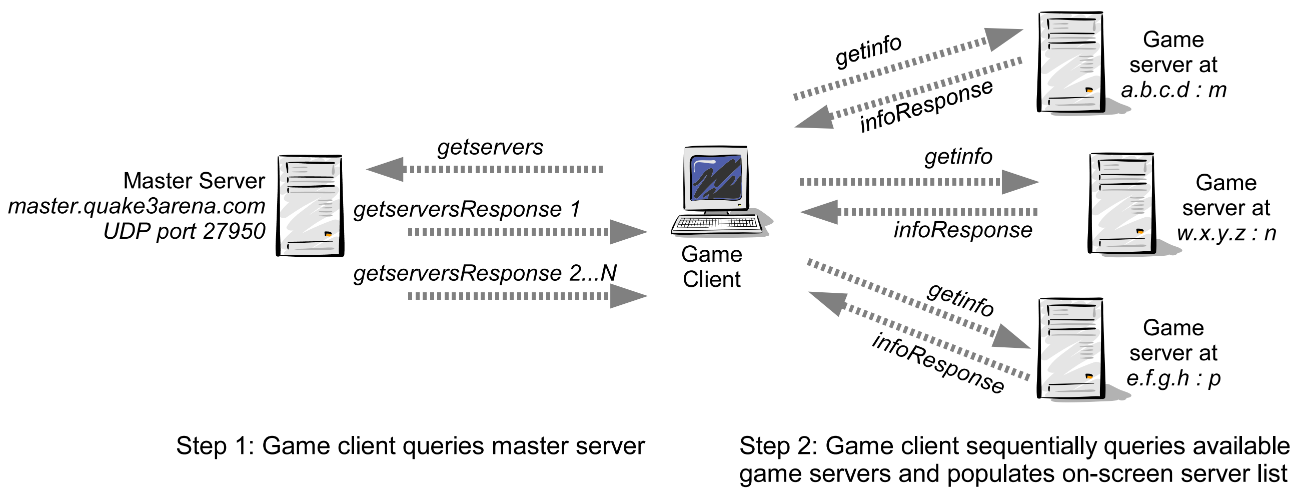
getinfo
infoResponse
Game server at a.b.c.d : m
getservers
Master Server
master.quake3arena.com
UDP port 27950
getserversResponse 1
getserversResponse 2...N
Game Client
getinfo
infoResponse
Game server at w.x.y.z : n
getinfo
infoResponse
Game server at e.f.g.h : p
Step 1: Game client queries master server
Step 2: Game client sequentially queries available game servers and populates on-screen server list

Fig. 1. A Quake III Arena client's discovery and probing of registered game servers

the perspective of Q3A, then the formats of command packets used during server discovery. This discussion assumes a Q3A client patched to version 1.32.

A. *Q3A server discovery sequence*

Public Q3A game servers automatically register themselves at *master.quake3arena.com:27950*, the Q3A master server. This master server becomes a rendezvous point for Q3A clients around the planet who wish to know what Q3A game servers are available at any point in time.

Figure 1 illustrates a Q3A client's server discovery process.

- The client sends a short *getservers* request packet to *master.quake3arena.com* on port 27950, eliciting one or more *getserversResponse* packets (typically well within 2 seconds). These *getserversResponse* packets contain all the currently registered Q3A game servers. (The Q3A master server returns multiple *getserversResponse* packets in quick succession when the list of registered game servers is too long to fit in a single UDP payload.)
- After *getserversResponse* packets are received the game client begins issuing *getinfo* probes to each listed game server. Game servers are probed in the order in which they were listed in the master server's *getserversResponse* packet(s).
- Each game server's reply comes back in an *infoResponse* packet. The game client populates its onscreen 'server browser' using information contained in each *infoResponse* packet and the game server's round trip time (RTT, estimated from the time between sending a *getinfo* and receiving a matching *infoResponse*).
- At any time during (or after) the *getinfo* / *infoResponse* process the player may chose a specific game server to play on from the information presented in the onscreen server browser.

From a Q3A game server's perspective, within minutes of registering with the master server it will begin seeing an influx of *getinfo* probe packets. These probes come from active Q3A clients around the Internet (Figure 2) and automated game server monitoring systems (such as ServerSpy [6]). The flow of player-triggered probe traffic will fluctuate with a 24-hour period [7], but rarely ever stop while the Q3A game server remains registered.

After establishing basic information about all registered game servers, Q3A clients may additionally issue a *getstatus* request to one or more selected game servers. This elicits a more comprehensive set of information about the game server in a *statusResponse* reply.

B. *Q3A server discovery packet formats*

Q3A uses UDP packets for all communication between master server, game servers and game clients. Command messages (such as the server discovery messages shown in Figure 1) are distinguished by their UDP payload beginning with 0xFFFFFFFF, indicating a Q3A out-of-band (OOB) command. The ASCII text following the 0xFFFFFFFF bytes indicates the command message itself. The UDP payload length indicates where the command message ends (no explicit termination bytes are used).

The following list describes the contents of the UDP

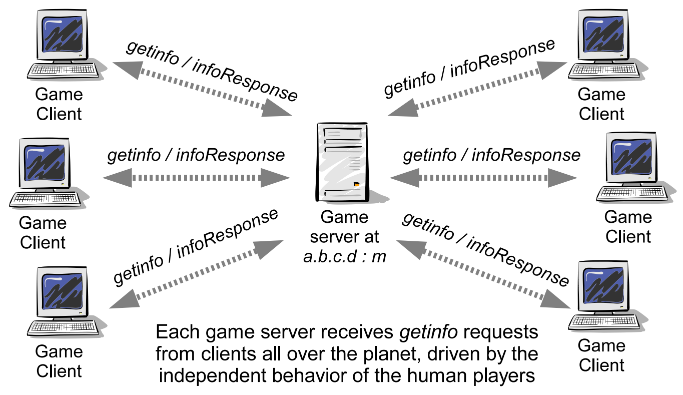

Fig. 2. Individual game servers experience *getinfo* probes coming from clients all around the Internet

payload after the 0xFFFFFFFF bytes for each of the OOB command packets in Figure 1.

- *getservers* packets consist of the ASCII text 'getservers NN' or 'getservers NN empty full'. The value NN indicates the network protocol level of the client (for example, a client patched to version 1.32 has a network code of 68). The master server returns only those game servers who previously registered with the same network protocol level. If the client's in-game server browser has "show empty" and "show full" set, the *getservers* string contains the additional words 'empty full'. (However, when tested in July 2007 it does not appear that the master server reacts any differently to these additional words being present.)
- *getserversResponse* packets consist of the ASCII text 'getserversResponse' immediately followed by a variable number of 7-byte fields and the trailing text '\EOT'. Each 7-byte field contains the character '\' (one byte), an IPv4 address (4 bytes) and UDP port number (2 bytes) of a registered game server. Each *getserversResponse* packet may carry up to 112 game servers (810 bytes of UDP payload per *getserversResponse* packet).
- *getinfo* packets consist of the ASCII text 'getinfo xxx'. 'xxx' is the challenge string - either literally three 'x' characters, or an arbitrary numberic string, that is returned as the challenge token value in subsequent *infoResponse* replies. (Alternatively, there may be no challenge string provided, and the UDP payload simply contains the ASCII text 'getinfo' and a trailing 0x0A byte.)
- *infoResponse* packets consist of the ASCII text 'infoResponse', a single byte 0x0A, and then a variable length ASCII 'infoString' carrying game-server specific details. The details (in '\token\value' form) include information such as the server's name, current map, current and maximum number players, etc. An example looks like:

  ```
  \game\osp\punkbuster\0
  \pure\1\gametype\0
  \sv_maxclients\20
  \clients\5\mapname\jof3dm1
  \hostname\EDs LAN Party
  \protocol\68\challenge\xxx
  ```

  (If the preceding *getinfo* packet lacked the ' xxx' then the *infoResponse* reply will lack the '\challenge\xxx' pair)
- *getstatus* packets consist of the ASCII text 'getstatus' terminated by a trailing 0x0A byte.
- *statusResponse* packets consist of the ASCII text 'statusResponse', a single byte 0x0A, and then a variable length ASCII infoString carrying additional game-server specific details (much more than in the *infoResponse* reply).

There are no sequence numbers in *getserversResponse* packets. The client simply collects and decodes all *getserversResponse* packets it receives after sending a *getservers* command to the master server.

Both *getinfo* and *getstatus* packets may contain an optional challenge string, which the probed game server should return as a '\challenge\xxx' pair in the infoString of the matching *infoResponse* or *statusResponse* reply.

Figure 3 illustrates the OOB packet format using an Ethernet frame carrying a *infoResponse*. The first byte in Figure 3 is the first byte of the Ethernet frame's header, and the UDP payload begins at offset 0x2A (42 bytes) from the Ethernet frame's start. This *infoResponse* was coming back from a game server at 68.88.65.185:1336 to a Q3A client at 10.1.1.2:27960 (the frame was captured behind a NAT box between the Q3A client and the Internet).

C. Q3A game server registration

Individual Q3A game servers intially register with the Q3A master server by sending a *heartbeat* packet to *master.quake3arena.com:27950*, and then repeating this *heartbeat* every 5 minutes while the game server is up. Figure 4 shows an Ethernet frame carrying a *heartbeat* packet from a public Q3A game server at 192.147.236.5.27960 to to the Q3A master server at 192.246.40.56.27950. The UDP payload contains the string 'heartbeat QuakeArena-1', terminated by 0x0A.

```
0000   00 16 e6 8c 0b 3a 00 12 bf 12 ab 7a 08 00 45 30  .....:.....z..E0
0010   00 b9 ae b6 00 00 75 11 05 3a 44 58 41 b9 0a 01  ......u..:DXA...
0020   01 02 05 38 6d 38 00 a5 0f 88 ff ff ff ff 69 6e  ...8m8........in
0030   66 6f 52 65 73 70 6f 6e 73 65 0a 5c 67 61 6d 65  foResponse.\game
0040   5c 63 70 6d 61 5c 70 75 6e 6b 62 75 73 74 65 72  \cpma\punkbuster
0050   5c 30 5c 70 75 72 65 5c 31 5c 67 61 6d 65 74 79  \0\pure\1\gamety
0060   70 65 5c 31 5c 73 76 5f 6d 61 78 63 6c 69 65 6e  pe\1\sv_maxclien
0070   74 73 5c 38 5c 63 6c 69 65 6e 74 73 5c 32 5c 6d  ts\8\clients\2\m
0080   61 70 6e 61 6d 65 5c 70 72 6f 2d 71 33 64 6d 36  apname\pro-q3dm6
0090   5c 68 6f 73 74 6e 61 6d 65 5c 57 65 6c 63 6f 6d  \hostname\Welcom
00a0   65 20 44 55 45 4c 20 53 65 72 76 65 72 5c 70 72  e DUEL Server\pr
00b0   6f 74 6f 63 6f 6c 5c 36 38 5c 63 68 61 6c 6c 65  otocol\68\challe
00c0   6e 67 65 5c 78 78 78                             nge\xxx
```

Fig. 3. A hex and ASCII dump of an Ethernet frame containing a Quake III Arena game server's *infoResponse* reply

```
0000  45 00 00 37 29 80 00 00 40 11 bb 6e c0 93 ec 05  E..7)...@..n....
0010  c0 f6 28 38 6d 38 6d 2e 00 23 47 a3 ff ff ff ff  ..(8m8m..#G.....
0020  68 65 61 72 74 62 65 61 74 20 51 75 61 6b 65 41  heartbeat.QuakeA
0030  72 65 6e 61 2d 31 0a                             rena-1.
```

Fig. 4. A hex and ASCII dump of an Ethernet frame containing a Quake III Arena game server's *heartbeat* registration packet

The Q3A master server derives each game server's IP address and port number from the source IP address and port number fields of the received *heartbeat* IP packet. This ensures the correct 'public' IP address and port information is recorded, even if the Q3A game server itself is located behind a NAT box.

While a Q3A game server is registered with the master server, another process running on *master.quake3arena.com* begins regularly probing the game server (acting like a Q3A client). Every 5 minutes a *getstatus* message is sent from *master.quake3arena.com* to the Q3A game server. The Q3A game server is then expected to respond with matching *statusResponse* message.

The infoString of each game server's *statusResponse* allows the master server to track that particular game server's network protocol level. This allows the master server to respond appropriately to *getservers* requests that specify a particular client network protocol level.

III. Wolfenstein Enemy Territory

ET was released in 2003 as an online-only team-play FPS game, and still has an active online player community. The ET server discovery sequence is very similar to that of Q3A (as the ET game engine is derived from Q3A code).

A. ET server discovery sequence

Public ET game servers automatically register themselves at *etmaster.idsoftware.com:27950*, the ET master server. This master server becomes a rendezvous point for ET clients around the planet who wish to know what ET game servers are available at any point in time.

The ET client's server discovery sequence is essentially the same as that described for Q3A and shown in Figure 1, with a few differences.

- The client sends a short *getservers* request packet to *etmaster.idsoftware.com* on port 27950, eliciting one or more *getserversResponse* packets within 2 seconds [8].
- As *getserversResponse* packets are received the game client begins issuing *getinfo* probes to each listed game server.

As a small optimisation, an ET client partially overlaps the reception of *getserversResponse* packets and the emission of *getinfo* probes. The first 16 game servers are probed in sequence as soon as the first *getserversResponse* packet arrives from the master server, with additional *getinfo* probes sent as previous probes are answered. No more than 16 probes remain outstanding (unanswered) at any one time.

From an ET game server's perspective, within minutes of registering with the master server it will begin seeing an influx of *getinfo* probe packets. These probes come from active ET clients around the Internet (Figure 2) and automated game server monitoring systems (such as ServerSpy [6]). The flow of player-triggered probe traffic will fluctuate with a 24-hour period [7], but rarely ever stop while the ET game server remains registered.

After establishing basic information about all registered game servers, the player may discover additional information about a particular game server by pressing the "Server information" button in the client's server selection browser. This triggers a *getstatus* request to the selected game server, eliciting additional information about the selected game server in a *statusResponse* reply.

B. ET server discovery packet formats

ET uses the same OOB control packet format as Q3A, starting each UDP packet with 4 bytes of 0xFFFFFFFF.

- *getservers* packets consist of the ASCII text 'getservers NN', where NN indicates the network protocol level of the client (for example, an ET client patched to version 2.60 has a network code of 84). The master server returns only those game servers who previously registered with the same network protocol level.
- *getserversResponse* packets consist of the ASCII text 'getserversResponse' immediately followed by a variable number of 7-byte fields and the trailing text '\EOT'. Each 7-byte field contains the character '\' (one byte), an IPv4 address (4 bytes) and UDP port number (2 bytes) of a registered game server. Each *getserversResponse* packet may carry up to 112 game servers.
- *getinfo* packets consist of the ASCII text 'getinfo xxx'.
- *infoResponse* packets consist of the ASCII text 'infoResponse', a single byte 0x0A, and then a variable length ASCII 'infoString' carrying game-server specific details. The details (in '\token\value' form) include information such as the server's name, current map, current and maximum number players, etc. An example looks like:

  ```
  \challenge\xxx\protocol\84
  \hostname\myServerName
  \serverload\0
  \mapname\fun_tennis
  \clients\0
  \sv_maxclients\10\gametype\2
  \pure\1\game\etpro
  \sv_allowAnonymous\0
  \friendlyFire\0
  \maxlives\0\needpass\0
  \punkbuster\1\gamename\et
  \g_antilag\1\weaponstrict\100
  \balancedteams\1
  ```
- *getstatus* packets consist of the ASCII text 'getstatus'.
- *statusResponse* packets consist of the ASCII text 'statusResponse', a single byte 0x0A, and then a variable length ASCII infoString carrying additional game-server specific details (much more than in the *infoResponse* reply).

There are no sequence numbers in *getserversResponse* packets. The client simply collects and decodes all *getserversResponse* packets it receives after sending a *getservers* command to the master server.

C. Other ET game client behaviour

The ET client (as of version 2.60) is a little more 'chatty' than the Q3A client on which it is based. Upon startup, and before the player has explicitly asked the client to do anything, the client is already passing information to online servers.

When the ET client first starts, it displays a splash screen and then shows the available options in the top left corner of the screen (to initiate play online, set local configuration options, etc). While in this mode, behind the scenes the client communicates with *etmaster.idsoftware.com:27951* (rather than the usual port 27950) to establish one's player number.

The sequence is:

- Client sends a *getmotd* OOB packet to *etmaster.idsoftware.com:27951*, consisting of the ASCII text 'getmotd ' (the trailing space is required) followed by a variable length ASCII infoString carrying some client-specific details.
- Server responds with a *motd* OOB packet, consisting of the ASCII text 'motd ' (the trailing space is required) followed by a variable length ASCII infoString carrying a 'message of the day' from the ET master server.

(Note: As with regular control packets, the above also begin the UDP payload with 0xFFFFFFFF.)

As of July 2007 a client *getmotd* message carries an infoString similar to this:

```
"\challenge\29862\renderer
\GeForce 7900 GS/PCI/SSE2
\version\ET 2.60 win-x86
 Mar 10 2005"\n
```

(In this case revealing to the master server information about my client's graphics card.)

The server's *motd* reply carries an infoString similar to this:

```
"challenge\29862
\motd\Welcome to Enemy Territory
 player 1025240255!!!\"
```

Note that in each case the quotation marks are explicitly included in the UDP payload. (Inspection of various control messages also suggests that a trailing 0x0A, rendered in ASCII as '\n', is optional - ASCII strings may end with 0x0A or simply end at the last byte in the UDP payload.)

Additionally, in July 2007 the version 2.60 ET client was observed resolving the IP address of *au2rtcw2.activision.com* and then sending a command message to *au2rtcw2.activision.com:27960* at the same time it was performing the *getmotd*/*motd* exchange. The client transmitted a *getUpdateInfo* OOB message, whose payload was:

```
getUpdateInfo "ET 2.60" "win-x86"\n
```

For unknown reasons the server at *au2rtcw2.activision.com:27960* was unavailable and the client received an ICMP 'port unreachable' message in response.

D. ET game server registration

Individual ET game servers intially register with the ET master server by sending a *heartbeat* packet to *etmaster.idsoftware.com:27950*, and then repeating this *heartbeat* every 5 minutes while the game server is up. Figure 5 shows an Ethernet frame carrying a *heartbeat* packet from a public ET game server at 136.186.229.32:27961 to the ET master server at 192.246.40.60:27950. The UDP payload contains the string 'heartbeat EnemyTerritory-1', terminated by 0x0A.

The ET master server derives each game server's IP address and port number from the source IP address and port number fields of the received *heartbeat* IP packet. This ensures the correct 'public' IP address and port information is recorded, even if the ET game server itself is located behind a NAT box.

While ever an ET game server is registered with the master server, another process running on *etmaster.idsoftware.com* begins regularly probing the game server (acting like an ET client). Every 4 minutes (or some integer multiple of 4 minutes) a sequence of *getchallenge*, *getstatus* and *getinfo* messages are sent (back to back) from *etmaster.idsoftware.com* to the ET game server. The ET game server is then expected to respond with matching *challengeResponse*, *statusResponse* and *infoResponse* messages.

The infoString of each game server's *infoResponse* allows the master server to track that particular game server's network protocol level. This allows the master server to respond appropriately to *getservers* requests that specify a particular client network protocol level.

(Note: the ET master server doesn't appear to care about ordering - *getstatus* is sent before *getinfo*, and the master server does not wait for a *statusResponse* before sending the *getinfo*.)

If the ET game server shuts down gracefully it will send an *ETFlatline* message to the master server. This is similar to a *heartbeat*, but the UDP payload contains the string 'heartbeat ETFlatline-1', terminated by 0x0A.

IV. QUAKE 4

Quake 4 (Q4) was released in October 2005 using an entirely new game engine based on Doom 3. Server discovery is conceptually similar to that used by Q3A, but Q4 introduces a new, incompatible packet format and some extra steps in the client's relationship to the Q4 master server.

First we will review the sequence of events from the perspective of Q4, then the formats of command packets used during server discovery. This discussion assumes a Q4 client patched to version 1.4.2.

A. Quake 4 server discovery sequence

Public Q4 game servers automatically register themselves at *q4master.idsoftware.com:27650*, the Q4 master server. This master server becomes a rendezvous point for Q4 clients around the planet who wish to know what Q4 game servers are available at any point in time.

A Q4 client's server discovery process comprises the following steps:

- The client sends a short *gameAuth* request packet to *q4master.idsoftware.com:27650*, carrying the client's 'CD Key' in clear text. This elicits an immediate *authKey* response from the master server.
- The client then sends a short *getServers* request packet to *q4master.idsoftware.com:27650*, eliciting one or more *servers* packets in reply (typically well within 2 seconds). These *servers* packets contain all the currently registered Q4 game servers. (The Q4 master server returns multiple *servers* packets in

```
0000   45 00 00 3b dd 98 00 00 40 11 46 0c 88 ba e5 20   E..;....@.F.....
0010   c0 f6 28 3c 6d 39 6d 2e 00 27 9c 48 ff ff ff ff   ..(<m9m..'.H....
0020   68 65 61 72 74 62 65 61 74 20 45 6e 65 6d 79 54   heartbeat.EnemyT
0030   65 72 72 69 74 6f 72 79 2d 31 0a                  erritory-1.
```

Fig. 5. A hex and ASCII dump of an Ethernet frame containing a Enemy Territory game server's *heartbeat* registration packet

quick succession when the list of registered game servers is too long to fit in a single UDP payload.)

- After *servers* packets are received the game client begins issuing *getInfo* probes to each listed game server. Game servers are probed in the order in which they were listed in the master server's *servers* packet(s).
- Each game server's reply comes back in an *infoResponse* packet. The game client populates its on-screen 'server browser' using information contained in each *infoResponse* packet and the game server's round trip time (RTT, estimated from the time between sending a *getInfo* and receiving a matching *infoResponse*).
- At any time during (or after) the *getInfo* / *infoResponse* process the player may chose a specific game server to play on from the information presented in the onscreen server browser.

Aside from the initial *gameAuth*/*authKey* sequence, the Q4 process looks broadly like the Q3A sequence shown in Figure 1.

As with Q3 and ET, the list of game servers returned by the master server may span multiple *servers* packets.

A Q4 client sends *getInfo* probes in bursts of 32 back to back packets, then pauses for a few hundred milliseconds before sending out another back to back burst of 32 *getInfo* probe packets.

The amount of information returned in *infoResponse* messages has grown (relative to Q3A and ET), and now includes active player details. This can cause the *infoResponse* message to be fragmented across two IP packets. (In such circumstances the fragmentation is performed at the IP layer, hidden from the Q4 client or server.)

B. Quake 4 server discovery packet formats

There are three main differences between Q4 and Q3A control messages: The UDP payload for OOB messages begins with 0xFFFF (rather than 0xFFFFFFFF), infoString token and value fields are deliminted by 0x00 (rather than '\') and there are some additional binary encoded fields (whose meanings are not precisely detailed in this report). Figures 6, 7 and 8 illustrate some of these differences.

Figure 6 shows a 63 byte Ethernet frame carrying a Q4 *getServers* request packet. This particular packet was coming from a Q4 client at 10.1.1.2:1590 to the master server at 192.246.40.28:27650.

Figure 7 shows the first 96 bytes and the final 62 bytes of a 1438 byte Ethernet frame carrying a Q4 *servers* reply packet. Up to 231 game servers may be returned in a single *servers* reply packet. The list of game servers begins at offset 0x34, immediately after the null-terminated 'servers' string.

This particular packet came from the Q4 master server at 192.246.40.28:27650 to the Q4 client at 10.1.1.2:1590, in response to the *getServers* request in Figure 6. The first listed game server is 85.236.101.43:28014 (starting at offset 0x34 the *55 ec 65 2b* represents the game server's IPv4 address and *6e 6d* indicates the game server's port). Additional game servers are listed in subsequent 6-byte fields.

Figure 8 shows the first 172 bytes and the final 55 bytes of a 919 byte Ethernet frame carrying a Q4 *infoResponse* reply packet. This particular packet was coming back from a Q4 game server at 69.65.15.4:28004 to a Q4 client at 10.1.1.2:1590.

C. Q4 game server registration

The process by which a Q4 game server registers with the Q4 master server is not covered by this report.

V. Conclusion

This report has provided some brief descriptions of the game server discovery sequence utilised by clients of Quake III Arena, Wolfenstein Enemy Territory and Quake 4 multiplayer online games. Discussion of the discovery sequence has been augmented by examples of actual packet payloads, sufficient for future developers to write their own packet parsing code.

References

[1] id Software, *Quake III Arena*, http://www.idsoftware.com/, as of July 21st 2007.

[2] ——, *Wolfenstein Enemy Territory*, under "Downloads" at http://www.enemyterritory.com/main.html, as of July 21st 2007.

```
0000   00 12 bf 12 ab 7a 00 16 e6 8c 0b 3a 08 00 45 00  .....z.....:..E.
0010   00 31 a2 25 00 00 80 11 a4 81 0a 01 01 02 c0 f6  .1.%............
0020   28 1c 06 36 6c 02 00 1d ef 0a ff ff 67 65 74 53  (..6l.......getS
0030   65 72 76 65 72 73 00 55 00 02 80 00 00 00 00     ervers.U.......
```

Fig. 6. A hex and ASCII dump of an Ethernet frame containing a Quake 4 client's *getServers* request

```
0000   00 16 e6 8c 0b 3a 00 12 bf 12 ab 7a 08 00 45 30  .....:.....z..E0
0010   05 90 00 00 40 00 36 11 4b 18 c0 f6 28 1c 0a 01  ....@.6.K...(...
0020   01 02 6c 02 06 36 05 7c 62 f1 ff ff 73 65 72 76  ..l..6.|b...serv
0030   65 72 73 00 55 ec 65 2b 6e 6d 55 ec 65 2b 64 6d  ers.U.e+nmU.e+dm
0040   d5 fb ad 20 c8 6d d5 fb ad 20 65 6d c2 74 52 05  ... .m... em.tR.
0050   64 6d 55 ec 64 3c 2c 6e 45 1c dc 03 64 6d 90 8c  dmU.d<,nE...dm..
[...lines elided for clarity...]
0560   c8 6d c3 7a 87 48 64 6d 55 15 25 13 ca 6d 08 09  .m.z.HdmU.%..m..
0570   03 9b 64 6d d5 72 b6 c7 64 6d c3 0d 3e 1e dc 69  ..dm.r..dm..>..i
0580   c3 0d 3e 12 94 75 c3 0d 3e 12 c8 6d 48 05 f9 2e  ..>..u..>..mH...
0590   64 6d c3 0d 3e 3a 08 6b c3 0d 3e 3a 48 71        dm..>:.k..>:Hq
```

Fig. 7. A hex and ASCII dump of an Ethernet frame containing a Quake 4 master server's *servers* reply

```
0000   00 16 e6 8c 0b 3a 00 12 bf 12 ab 7a 08 00 45 30  .....:.....z..E0
0010   03 89 3d f8 00 00 77 11 a2 f4 45 41 0f 04 0a 01  ..=...w...EA....
0020   01 02 6d 64 06 36 03 75 8a 16 ff ff 69 6e 66 6f  ..md.6.u....info
0030   52 65 73 70 6f 6e 73 65 00 01 00 00 00 55 00 02  Response.....U..
0040   00 6e 65 74 5f 73 65 72 76 65 72 44 65 64 69 63  .net_serverDedic
0050   61 74 65 64 00 31 00 66 73 5f 67 61 6d 65 5f 62  ated.1.fs_game_b
0060   61 73 65 00 00 66 73 5f 67 61 6d 65 00 71 34 6d  ase..fs_game.q4m
0070   70 00 73 76 5f 70 75 6e 6b 62 75 73 74 65 72 00  p.sv_punkbuster.
0080   31 00 73 69 5f 76 65 72 73 69 6f 6e 00 51 75 61  1.si_version.Qua
0090   6b 65 34 20 20 56 31 2e 34 2e 32 20 77 69 6e 2d  ke4  V1.4.2 win-
00a0   78 38 36 20 4a 75 6e 20 31 35 20 32 30 30 37 00  x86 Jun 15 2007.
[...lines elided for clarity...]
0360   73 69 5f 6e 75 6d 50 72 69 76 61 74 65 50 6c 61  si_numPrivatePla
0370   79 65 72 73 00 30 00 67 61 6d 65 6e 61 6d 65 00  yers.0.gamename.
0380   51 34 4d 50 20 31 2e 34 2e 32 00 00 00 20 07 00  Q4MP 1.4.2... ..
0390   00 00 00 00 00 00 01                             .......
```

Fig. 8. A hex and ASCII dump of an Ethernet frame containing a Quake 4 game server's *infoResponse* reply

[3] (fan site), *Wolfenstein Enemy Territory*, http://www.enemy-territory.com/, as of July 21st 2007.

[4] id Software, *Quake 4*, http://www.quake4game.com/, as of July 21st 2007.

[5] *ioquake3Wiki*, http://hobbshouse.org/wiki/index.php/Main_Page, as of July 21st 2007.

[6] *ServerSpy - Online PC Gaming Statistics*, http://www.serverspy.net/, as of July 21st 2007.

[7] S. Zander, D. Kennedy, and G. Armitage, "Dissecting server-discovery traffic patterns generated by multiplayer first person shooter games," in *Proceedings of ACM Networks and System Support for Games (NetGames) Workshop*, October 2005.

[8] G. Armitage, C. Javier, and S. Zander, "Measuring a wolfenstein enemy territory master servers response to game client queries," CAIA Technical Report 060410A, http://caia.swin.edu.au/reports/060410A/CAIA-TR-060410A.pdf, April 2006.