Observations on game server discovery mechanisms

Tristan Henderson*
Department of Computer Science
University College London
Gower Street
London WC1E 6BT, UK
T.Henderson@cs.ucl.ac.uk

ABSTRACT

Networked First Person Shooter (FPS) games are amongst the most popular multiuser applications on the Internet today. At any given time, there are thousands of servers available to a potential player. We describe and analyse the existing mechanisms for locating these game servers. The mechanisms are found to be inefficient and do not scale well. We propose and describe a distributed peer-to-peer server discovery mechanism. This is a work in progress.

Categories and Subject Descriptors

C.2.4 [**Computer-Communication Networks**]: Distributed Systems—*distributed applications*

General Terms

Measurement

Keywords

games, measurement, discovery

1. INTRODUCTION

First person shooter (FPS) games such as *Quake* and *Half-Life* represent one of the most popular multiuser applications on the Internet today. These games use a client-server architecture, with a large number of servers each providing a game for a relatively small number of players (typically there are between 16-32 players). At any given time, there can be several thousand servers available to a potential player. The mechanisms for locating these servers are very basic and inefficient. In this paper we analyse the current mechanisms for server discovery, and suggest some ways in which these could be improved.

This paper is structured as follows. In Section 2 we describe the existing methods of locating a game server. Section 3 describes the queries that we have carried out to analyse the performance of these existing mechanisms, and the results of these queries. In Section 4 we propose and describe a distributed server discovery mechanism. Finally, in Section 5 we outline directions for future work.

*Tristan Henderson is funded by an EPSRC CASE award in conjunction with HP Labs.

NetGames2002 April 16-17 2002, Braunschweig, Germany

2. EXISTING SERVER DISCOVERY MECHANISMS

When a user decides to join an FPS game on the Internet, they need to know the location of an appropriate game server to which they can connect. This location comprises the IP address and the port number on which a game server is running, and in this paper we will refer to this address/port pair as a **server address**. This server address can be obtained out-of-band, for example from a friend who is currently playing in a session, or from prior knowledge, for instance if they always tend to play on the same server. If no server addresses are known to the potential player, then the player can query a **server directory**[1] to obtain the location of a suitable game server.

The server directory query mechanism for the game *Half-Life* [20] operates as follows:

- When a game server starts running, it registers with one or more server directories. The addresses of these directories are discovered by the server operator out-of-band and the server program is configured appropriately.
- A potential game player sends a 6 byte UDP packet containing the character "e", followed by a sequence number indicating where in the list the server directory should start returning addresses. For the initial query, this number should be 0.
- The server directory responds with a UDP packet of up to 1396 bytes long. This contains a one byte sequence number, followed by a list of server addresses.
- If the sequence number returned by the server directory is non-zero, the client sends another query packet to the server directory, this time including the returned

[1]In FPS games these are typically referred to as "master servers" — however, to reduce confusion with the term "game server", we refer to the master server as a "server directory" in this paper.

sequence number. The server directory then responds with another list of server addresses, until the client has received the entire list.

We have analysed several of the most popular FPS games: *Half-Life*, *Hexen*, *Heretic*, *QuakeWorld*, *Quake II*, *Quake III: Arena*, *Return to Castle Wolfenstein*, and *Unreal: Tournament*. The general server discovery procedure described above is the same for all of these FPS games, except that several games do not transmit sequence numbers. This means that in the event of packet loss, the entire list has to be retransmitted, since UDP is being used. *Half-Life* also appears to be unique in that a client has to authenticate with the server directory using a challenge-response system — according to the game's developers, this is intended to prevent denial of service attacks [3].

Once the user has obtained this list of game servers, they are then able to query each of the servers in the list until they find a game that they would like to join. Most FPS game servers offer a query mechanism whereby such details as the number of players, the current game in progress, the players' scores and so on can be retrieved. This information allows the player to decide whether or not to join a particular server. Like the server directory query mechanism, the game servers are queried using UDP.

3. ANALYSIS

We have written a set of Perl modules to query both the game servers and server directories for the FPS games mentioned in the aforementioned list.

3.1 Methodology

The primary *Half-Life* server directories are those provided by `won.net`, the "World Opponent Network". There are three such servers, `half-life.east.won.net`, `half-life.central.won.net` and `half-life.west.won.net`, which are designed to service the appropriate regions of the USA. Thus, a player located in California can query `half-life.west.won.net` and find all the game servers that wish to advertise to players on the American west coast. In addition to the primary server directories, we compiled a list of five other server directories from various gaming websites. These included server directories oriented towards non-American players, and server directories run by individual game server operators for their own purposes. Each of the server directories was queried four times a day for a month. For each query, we recorded the list of game servers returned by the server directory, and the number of times each server appeared in the list. We also recorded the number of duplicate packets sent by the server, and the number of retransmitted packets (if any) required to be sent by the client (e.g., in the event of packet loss or the server timing out). Having obtained a list of all the available game servers, we then sent a query packet to each server. Ten servers were queried at a time, and a server was assumed to be inactive if no response was received after ten seconds. From analysing network traffic, we believe that this is how the *Half-Life* client operates.

3.2 Results

Table 1 lists a summary of our observations. We have identified several problems with the current server directory system.

Single point of failure The most obvious problem with a centralised server directory is that if that server is down, then potential players are unable to find any game servers. This was not a problem for the main servers at `won.net` during our polling, and they were always reachable. Some of the other, smaller, server directories, however, were frequently inactive. This reachability problem is compounded in *Half-Life*, since players also have to identify themselves with a centralised authentication server before they can connect to a game server. On the other hand, one advantage of a single, central server is that it is easy to charge for access to that information, and some companies, e.g. GameSpy [9] have set up their own server directories to do so.

Information is stale The list of server addresses that is returned by the server directories does not appear to be very reliable, and on average, only 31.32% of the game servers advertised by the server directory were found to be active. This might be due to a number of factors. Many servers are run by residential users, who many have dynamically allocated IP addresses. Such residential servers are also more likely to be behind a NAT, which, in the event of a server restart or crash, may increment the port number used by the server. Thus, a server address might not be very permanent, and so might have changed before the server has been able to update its index. Since, however, each server has to re-register with the server directory every three minutes, this should eliminate many of the stale server addresses, and so it is likely that there is a bug in the server directory program.

Information is redundant As we have already mentioned, `won.net` provides three server directories to service the relevant regions of the USA. However, of these three server directories, 87.77% of the total set of servers observed appeared in all three server directory lists. In other words, most server operators tend to register with all three server directories. Whilst redundancy can be seen as a benefit, for instance to increase resilience and avoid the problems associated with a single point of failure, in this case it only creates additional problems. All three `won.net` servers appear to be on the same subnet, and so it is likely that a connectivity failure would result in all three servers being unavailable. Since most servers advertise on all three server directories, the geographic relevance implied in the server names becomes ineffective. A player located in the western part of the USA cannot trust that all the servers on `half-life.west.won.net` are located near to them. Thus, the outcome is that a player may have to query all three server directories in order to get an overall picture of the servers available to them. Some clients do not allow users to specify which server directory they wish to contact (for instance, making assumptions about location based on hostname and automatically contacting the "closest" server directory), and this may make any search results even less reliable.

Table 1: Summary of observations

	Min	*Max*	*Mean*	*Median*
Number of advertised servers	7984	20930	18891.10	18789
Number of active servers	2458	7588	5659.08	5804
Active servers (as % of advertised)	16.32	72.19	30.21	30.71
Time taken for query (seconds)	310	676	656.45	670

There is also some redundant information within a single server. On average, 4.4% of the servers on each of the `won.net` server directories is listed more than once. We suspect that this is also due to a bug in the server directory program.

Affected by network conditions Since all of the server queries use UDP, none of the congestion control mechanisms associated with TCP are available to the servers. Thus, an important issue is how well the server directories can deal with adverse network conditions. The server directory lists at the main `won.net` servers comprised an average of 115.82 Kb of data, which was transmitted in an average of 86.39 packets (since UDP is used, path MTU discovery [12] is difficult, and thus the server directory always sends packets of 1396 bytes in size). Whilst we rarely saw loss in our observations, when there was loss, the loss rate was quite high — an average of 15 packets or 17.36%. This is because the clients do not back-off; instead, they request the packets which have been lost at a constant rate, aggravating the conditions which are causing the existing packet loss.

No standardised interface The server directory mechanism that we analyse here is the default method for advertising the presence of a *Half-Life* server. By this we mean that this mechanism is built into the server program itself, and no additional software is required for a game server operator to advertise with the server directory. We have found, however, that game server operators have also chosen to use several other mechanisms. Several operators advertise their servers on their WWW pages, for example. Others allow users to finger their servers to query availability. These other interfaces may have been implemented to draw attention to other information, such as advertising and so on, or perhaps due to the unreliable nature of the server directory mechanism. Whilst this is not a design problem *per se*, a more reliable server discovery mechanism that was standardised for a number of games, might reduce this variation in interfaces. Some server directory operators have already taken matters into their own hands, and have chosen to use the same server directory program for a variety of games, such that some *Half-Life* server directories do not respond to a *Half-Life* server query, but do respond to a *Quake III* query.

One size fits all The server directory returns a list of all the servers that have chosen to register with it. As we have already mentioned, a server is free to register with as many server directories as they wish. The list returned by the server directory only contains raw server addresses, and a prospective player then has to query all the servers in turn, since there is no additional information about the servers by which to filter and indicate interest. This query process can therefore take a long time, since a client might be querying servers on the other side of the world to which they have no intention of connecting. In our observations the queries took an average of almost 11 minutes.

4. A PEER-TO-PEER SERVER DISCOVERY SYSTEM

The server directory mechanism that we have described here is very similar to the original Napster peer-to-peer file sharing system. A client who wishes to retrieve a given file *song.mp*3 from the Napster system connects to a central Napster server, typically the main server at `http://www.napster.com`. After authentication, the client uploads a list of files that they wish to share with the other users of the system. The central server updates the index, and the client can then query the server, which responds with a list of IP addresses of the Napster clients that have *song.mp*3 available for download. The client then chooses one of these IP addresses and connects directly to that peer to retrieve *song.mp*3. The central Napster server is thus almost identical to the game server directory — they both maintain a central index which is used by all the Napster clients, or game servers.

Given the similarities between server directories and Napster, we believe that it is beneficial to learn from the research in peer-to-peer networking when developing new server discovery mechanisms. Partly as a result of the success of systems such as Napster, KaZaA and Gnutella, peer-to-peer networks have become a highly active area of research. Many of the current systems, however, such as Chord [19] and CAN [15] assume that there is a single instance of each datum in the network. In the Freenet network [4], for instance, search queries pass from node to node until the first successful response is received, and then the data is passed back to the querier. The same function is used for searching and inserting, and an insert can only take place if a search proves unsuccessful. It is therefore impossible to have more than one copy of each piece of data, and indeed this is a design feature, since it prevents a document from being retracted or deleted by overwriting it. Systems which do allow multiple copies of data, such as application-level anycast [7], often assume that only one copy, such as the closest or most responsive, is required by the client requesting the data. Other systems such as the Eternity service [2] and Publius [22] also allow multiple copies of data for resistance to censorship, but again each copy is identical. These systems are not really suited to discovering game servers, where a list of positive query results is required, from which the client can make a decision which might be based on non-network

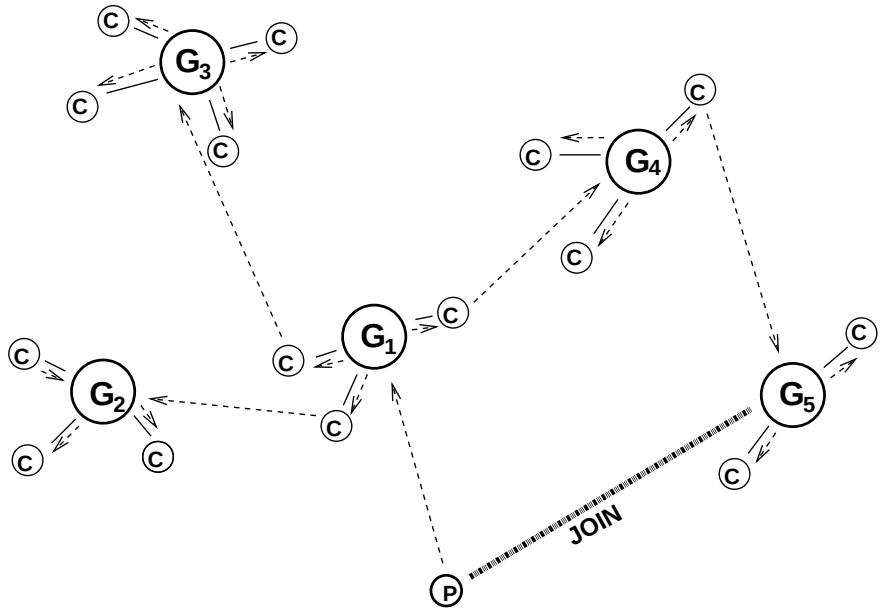

Figure 1: Locating and selecting a game server

factors, such as who else is playing on the server. In a server discovery system, the aim is therefore not just a lookup, but a pair of goals: to obtain a list of server locations, and then to select the best location from that list.

Decentralised systems where multiple resources can be located, however, do exist, and have done for some time, for instance the Grapevine system [18], and more recent gossip-based systems such as Captain Cook [21]. Gossip-based systems are analogous to an epidemic [5]. Nodes randomly choose other nodes to "gossip" with, that is, to spread information to, such as a query or a database update.

We are currently designing a distributed game server discovery system. The system takes advantage of the existing game servers and the players that connect to these servers, and use these as the nodes in an application-level network. Figure 1 shows the general idea of our system. Each game server caches the addresses of the players who have most recently played on it, and each player caches the addresses of the servers on which it has most recently played. A potential player who wishes to find a server then sends a "gossip" message to one of the servers in its cache. That server then relays the message to the players in its cache, who then relay onto the servers in their caches, and so on. At each additional hop, if suitable servers are found, their addresses are relayed back to the original querier. Each new server address is queried, and once a suitable server has been found, the player connects to that server and the "gossip" terminates. In Figure 1 the potential player P sends a message to G_1. Messages pass via the various clients (marked C) from G_2 to G_4, until finally G_5 is reached. P finds G_5 suitable, and so connects to it.

The advantage of a distributed discovery system is that it can resolve some of the problems described in section 3.2. A decentralised system has no single point of failure. By querying a server that has been recently used, a player is more likely to receive results from the servers closer to them, and so the problem of querying distant servers is eliminated. Furthermore, by caching previously used servers and players, users are more likely to connect to servers to which they have some sort of social bond; for instance, where their friends are playing, which is a phenomenon which we have already observed in practice [10]. This may be especially important in team-based games such as the highly popular *Half-Life* modification *CounterStrike* [11].

4.1 Potential problems

There are many issues that need to be considered when designing any peer-to-peer system. The "tragedy of the commons" has been observed in peer-to-peer systems [1, 17] — users are quite happy to take information from others, but are not always so happy to share data. In a server discovery context, a user might not be willing to indicate which servers they have played on. This would greatly affect the efficiency of the system as a whole. However, because of the multiuser nature of an FPS application, it should be easy to discover misbehaving nodes, since they will be visible to other players. This information could be used to enforce sharing, such as in the DirectConnect file-sharing network [6], and users who refuse to route server queries could be banned from joining servers in the same way that players who currently cheat in games tend to be banned.

One common problem in decentralised resource location systems is keeping the data store of location information current, and maintaining consistency between nodes. If, however, players tend to connect to servers closest to them, keeping a global database might not be required. Instead, it is only necessary to ensure that the location data concerning the nearest servers is current. This will be dependent on how active the nearby players are.

The scalability of a system where each node passes on successful queries, as well as initiating further queries, might be limited. Indeed, the lack of a scalable query mechanism has been one of the large stumbling blocks for the Gnutella file-sharing network [16].

Perhaps the most difficult problem for peer-to-peer networks to overcome is how to bootstrap a system. How does a new node enter the system, and how do you find out where the other nodes are? We envisage that our system will be used in conjunction with the existing server directory mechanism, and a new player who has never played the game before can either query the server directory, find a node out-of-band, or start their own server. Host caches, such as in the Gnutella network, where popular and relatively permanent nodes are listed on websites, are another possible solution. Unfortunately, all these measures bring some element of centralisation to the decentralised network, but we do not know of any peer-to-peer system that has yet to satisfactorily resolve this issue.

5. CONCLUSIONS AND FUTURE WORK

The client-server architecture is currently the most popular choice for implementing multiplayer networked first person shooter games. This is unlikely to change in the near future, since games developers appear to prefer a central server due to advantages such as ease of programming [8, 13]. Using a client-server architecture for small-scale games such as First Person Shooters will thus lead to a proliferation of servers for players to choose from. This paper has described and analysed the existing mechanisms for discovering FPS game servers on the Internet.

Programming bugs aside, many of the problems that we have found can be attributed to the choice of UDP as a transport protocol. According to games developers, UDP was chosen because of the overhead of creating connections in TCP [3]. As can be expected, however, the unreliable nature of UDP and created its own problems, which we have described here. Games developers might thus wish to consider the use of TCP in future games for non-realtime communication such as server discovery.

As an alternative to the centralised server directory, we have outlined our plans for a distributed peer-to-peer discovery mechanism. Much design work is required before we can implement the system. We have indicated some of the potential problems with our system, and we hope to discuss these and other aspects at the workshop.

The experiments that we have described here are limited in that they originated from a single site at UCL. We hope to replicate the queries from multiple sites in the future.

The efficiency of a distributed discovery mechanism will be greatly impacted by the longevity of the games servers - if their addresses and lifespan are highly variable, it will be difficult to discover them through distributed means. We intend to run some server polls to estimate the lifespan of existing *Half-Life* servers.

6. ACKNOWLEDGEMENTS

Thanks to Jon Crowcroft and the reviewers for their many useful comments. Steve Jankowski's QStat tool [14] was invaluable in understanding some of the inner workings of game server query protocols.

7. REFERENCES

[1] E. Adar and B. A. Huberman. Free Riding on Gnutella. *First Monday*, 5(10), Oct. 2000.

[2] R. Anderson. The Eternity service. In *Proceedings of Pragocrypt '96*, pages 242–252, Prague, Czech Republic, Sept. 1996.

[3] Y. W. Bernier. Half-Life and TeamFortress networking: Closing the loop on scalable network gaming backend services. In *Proceedings of the 14th Games Developers Conference*, San Jose, CA, Mar. 2000.

[4] I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong. Freenet: A distributed anonymous information storage and retrieval system. In *Proceedings of the International Workshop on Design Issues in Anonymity and Unobservability*, Berkeley, CA, July 2000.

[5] A. Demers, D. Greene, C. Hauser, W. Irish, J. Larson, S. Shenker, H. Sturgis, D. Swinchart, and D. Terry. Epidemic algorithms for replicated database maintenance. In *Proceedings of the Sixth Annual ACM Symposium on Principles of distributed computing*, pages 1–12, Vancouver, Canada, Aug. 1987.

[6] Direct Connect file-sharing network. `http://www.neo-modus.com`.

[7] Z.-M. Fei, S. Bhattacharjee, E. W. Zegura, and M. Ammar. A novel server selection technique for improving the response time of a replicated service. In *Proceedings of the 17th IEEE Conference on Computer Communications (INFOCOM)*, pages 783–791, San Francisco, CA, Mar. 1998.

[8] T. Funkhouser. Network topologies for scalable multi-user virtual environments. In *Proceedings of the Virtual Reality Annual International Symposium 1996 (VRAIS '96)*, pages 222–229, Santa Clara, CA, Apr. 1996.

[9] GameSpy. `http://www.gamespy3d.com`.

[10] T. Henderson. Latency and user behaviour on a multiplayer game server. In *Proceedings of the 3rd International Workshop on Networked Group Communication (NGC)*, pages 1–13, London, UK, Nov. 2001.

[11] T. Manninen. Virtual Team Interactions in Networked Multimedia Games — Case: "Counter-Strike" — Multi-player 3D Action Game. In *Proceedings of the 4th Annual International Workshop on Presence (PRESENCE 2001)*, Philadephia, PA, May 2001.

[12] J. Mogul and S. Deering. Path MTU Discovery, Nov. 1990. RFC 1191.

[13] Y.-S. Ng. Internet game design. *Gamasutra*, Aug. 01, 1997. `http://www.gamasutra.com/features/19970801/ng.htm`.

[14] QStat. `http://www.qstat.org`.

[15] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. In *Proceedings of ACM SIGCOMM 2001*, pages 161–172, San Diego, CA, Aug. 2001.

[16] J. Ritter. Why Gnutella Can't Scale. No, Really., 2001. `http://www.darkridge.com/~jpr5/doc/gnutella.html`.

[17] S. Saroiu, P. K. Gummadi, and S. D. Gribble. A measurement study of peer-to-peer file sharing systems. In *Proceedings of Multimedia Computing and Networking 2002 (MMCN '02)*, San Jose, CA, Jan. 2002.

[18] M. D. Schroeder, A. D. Birrell, and R. M. Needham. Experience with Grapevine: The growth of a distributed system. *ACM Transactions on Computer Systems*, 2(1):3–23, Feb. 1984.

[19] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: a scalable peer-to-peer lookup service for Internet applications. In *Proceedings of ACM SIGCOMM 2001*, pages 149–160, San Diego, CA, Aug. 2001.

[20] Valve Software. Half-Life. `http://www.sierrastudios.com/games/half-life/`.

[21] R. van Renesse. Scalable and secure resource location. In *Proceedings of the 33rd Hawaii International Conference on System Sciences*, pages 1209–1218, Maui, HI, Jan. 2000.

[22] M. Waldman, A. D. Rubin, and L. F. Cranor. Publius: a robust, tamper-evident, censorship-resistant, web publishing system. In *Proceedings of the 9th USENIX Security Symposium*, pages 59–72, Denver, CO, Aug. 2000.