Discovering First Person Shooter game servers online: techniques and challenges

Grenville Armitage

Center for Advanced Internet Architectures,
Swinburne University of Technology,
Melbourne, Australia
E-mail: garmitage@swin.edu.au

Abstract: Multiplayer online first person shooter games are usually client-server based. This paper reviews networking issues and tradeoffs associated with common techniques used to discover playable game servers. ISPs, game hosting companies and private enthusiasts typically host game servers and publishers host 'master servers'. To seek available game servers clients query the master servers for lists of currently active game servers then probe the list for status information and latency estimates. Slow probing irritates players, whilst probing too quickly congests consumer's links (inflating latency estimates). A game server's inbound probe traffic reveals the topological distributions of likely player populations.

Keywords: online games; server discovery; FPS; first person shooter; consumer networking.

Reference to this paper should be made as follows: Armitage, G. (2008) 'Discovering First Person Shooter game servers online: techniques and challenges', *Int. J. Advanced Media and Communication*, Vol. 2, No. 4, pp.402–414.

Biographical notes: Grenville Armitage earned a BEng (Elec) (Hons) in 1988 and a PhD in Electronic Engineering in 1994, both from the University of Melbourne, Australia. Since 2002 he has been an Associate Professor of Telecommunications Engineering and Director of the Centre for Advanced Internet Architectures at Swinburne University of Technology. He authored Quality of Service in IP Networks: Foundations for a Multi-Service Internet (Macmillan, April 2000) and co-authored Networking and Online Games – Understanding and Engineering Multiplayer Internet Games (Wiley, April 2006). He is a member of IEEE, ACM and ACM SIGCOMM.

1 Introduction

Internet-based multiplayer First Person Shooter (FPS) games (such as Quake III Arena, Wolfenstein Enemy Territory, Half-Life 2, Counter-Strike: Source, Unreal Tournament and Enemy Territory Quake Wars) have become quite common in the past 6+ years. FPS games typically operate in a client-server mode – each player controls

a client (on a PC or dedicated games console) that communicates with a central *game server* hosting the game itself. Game servers are hosted by Internet Service Providers (ISPs), dedicated game hosting companies and individual enthusiasts. Although most individual FPS game servers only host from 4 to 30+ players, there are usually many thousands of individually operated game servers active on the internet at any given time (Armitage et al., 2006; Feng and Feng, 2003).

The challenge for game clients is to locate up-to-date information about the game servers available at any given time, such that a prospective player can select a suitable server on which to play. This process is termed *server discovery*, and is usually triggered manually by the human player to populate or refresh a list of available servers presented by their client's on-screen 'server browser'. The challenge for game designers is to make server discovery prompt, accurate while using minimal network resources.

This paper reviews the networking issues and tradeoffs associated with common techniques used for FPS server discovery. We begin with the reasons behind FPS games using client-server communication, summarise the server discovery problem and use Counter-Strike: Source (CS:S) and Wolfenstein Enemy Territory (ET) to illustrate some specific issues. We discuss the impact of server discovery traffic on game servers 24-hours per day, the importance of not sending probes too fast, and the impact a master server's ranking scheme has on the visibility of individual game servers. Finally we note a number of possible optimisations to reduce probe traffic and create a more timely discovery process.

2 Background

Multiplayer online games tackle a number of issues when chosing their network communication model. A player's perception of the virtual game environment depends on regular and timely game-state updates indicating the actions of other players and the consequences of the local player's own actions. FPS games tend to utilise a client-server communication model because, relative to peer to peer models, network traffic patterns are simplified and the game server provides a nominally trusted independent entity to impose the game's rules and minimise opportunities for cheating.

2.1 The roles of clients and servers

A game server receives frequent update (*command*) messages from all clients (indicating movements and actions performed by an individual player) and sends regular updates (*snapshots*) to all clients (indicating what has occurred inside the virtual world since the server's previous update). The game server decides when players have successfully performed actions that score points (such as shooting another player), and ensures all player characters obey whatever 'laws of physics' apply to all players inside the virtual world. A game server never reveals client IP addresses to other clients, minimising the likelihood of one upset player directing an IP-layer denial of service attack at another player's network connection.

Messages are typically sent using UDP (rather than TCP) over IP because a real-time immersive game experience requires timely data transfer more so than reliable data transfer. A game server also updates each client only with information regarding events that should be visible to that client's player. This limits opportunities for

client-side cheats (such as revealing other players hiding behind walls) and helps minimise the size of snapshot packets sent from the server to each client.

The client-server model also minimises the number of UDP/IP <address:port> pairs that must be handled during active game-play by NAT-enabled consumer routers (increasingly prevalent between game clients and the internet).

2.2 Establishment and discovery of FPS game servers

Most PC-based FPS games with an online multiplayer mode are published as a commercial game client and a matching, freely distributed game server. The online FPS client is often bundled as part of a standalone offline game, or released as a separate 'mod' (modification to the game). FPS game publishers then rely on communities of enthusiasts (ISPs, game hosting companies and individuals) to host hundreds or thousands of independent game servers all over the internet. (Game server functionality may also be built into the game client code – sometimes referred to as a 'listen server'. One player uses their game client to also provide the game server for a group of friends.)

FPS game servers typically host less than $\sim$60 players, largely because the CPU and network requirements to support more players tends to conflict with the 'rely on enthusiasts' model of free hosting.

A key challenge for FPS game clients is using minimal network traffic while promptly identifying remote game servers by player-relevant attributes (such as the number of current players on each game server and the network latency to each game server from each client's perspective).

In lieu of hosting their own game servers, FPS publishers typically provide a form of rendezvous service. The most common approach is to provide *master servers* specific to each FPS game (Henderson, 2002). Individual game servers register themselves with one or more master servers when they start up. FPS clients know the identity of one (or a small number of) master servers. When directed by a human player, a game client queries their master server for the IP address and port numbers of all currently active game servers. The client then performs server discovery – probing all the registered game servers in order to present the player with a list of available game servers and their relevant characteristics. Thousands of probes may be sent by a client before the player selects a single game server on which to play.

2.3 Alternative schemes

More tightly integrated approaches have been recently deployed in the console market. For example, Halo 3 on XBox Live utilises a slightly different rendezvous service (Lee et al., 2008). XBox Live identifies groups of clients who wish to play against each other and designates one of those client consoles to (temporarily) act as the game server for everyone else in the group. This approach relies on the console's own security model to assure the integrity of the nominated game server.

Other schemes have proposed to subsequently redirect clients from one game server to another based on certain metrics intended to improve the game-play experience (such as load-balancing by geography (Chambers et al., 2003)). However, such augmentations are (as with the XBox Live technique) more about game server *allocation* than game server *discovery* and so we will not address them further in this paper.

Distributed server discovery schemes have been proposed for traditional FPS games, often to eliminate the master servers as a potential single point of failure (such as Henderson, 2002). However, a key weakness tends to be their assumption that thousands of players and/or independent game server operators will cooperate by honestly sharing queries or information about other game servers they have seen recently. In reality both players and game server operators have relationships that range from neutral to potentially adversarial. (Players may wish other players to not find 'good' servers, and game server operators may attain higher status by influencing more potential players to their own servers rather than someone else's game servers.)

In this paper we focus on the issues and trade-offs associated with using third-party master server(s) to identify active game servers, then relying on game clients to make their own direct enquiries of available game servers.

3 Illustrating FPS server discovery

Players cannot, a priori, know how many game servers are available, what virtual world ('map') or game each server is offering, which servers actually have any other players on them, and the latency (or round trip time, RTT) between the client and each server. Latency is an important selection criteria because competitive online FPS game play typically requires latencies below 150–200 ms (Armitage et al., 2006). Latency to different game servers must be measured relative to each client's location. Consequently, FPS server discovery uses short request-response probe pairs to estimate latency and collect game server-specific information of interest to the player.

We will illustrate a number of network-layer design decisions relating to server discovery using examples based on two different online FPS games – Counter-Strike: Source (CS:S) and Wolfenstein Enemy Territory (ET). Modulo small design differences, the issues covered here apply to server discovery for all FPS games that use distributed, enthusiast-hosted game servers.

3.1 Counter-Strike: Source

Counter-Strike: Source (CS:S) is a 'mod' for the Half-Life 2 game engine, and was first released in late 2004 by Valve Corporation through their Steam online game delivery system. An extremely popular team-based FPS, in early 2008 there were over 30,000 public CS:S game servers registered with Steam's master server at *hl2master.steampowered.com*. Figure 1 illustrates the steps that occur when a player initiates server discovery through their Steam client's server browser.

A Steam client will:

- Issue a *getservers* query packet to UDP port 27011 on the Steam master server.
- Receive a *getserversResponse* packet containing the <IP address:port> pairs of up to 231 active game servers.
    - Send one *A2S_INFO Request* probe packet to each of the 231 game servers in order, eliciting an *A2S_INFO Reply* packet from every active game server.

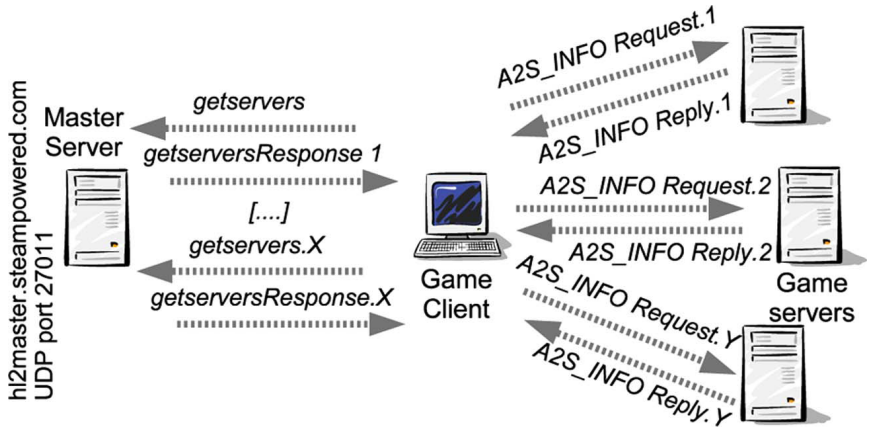
- Update the on-screen server browser with details from each *A2S_INFO Reply* (with an RTT estimated from the time between sending a *A2S_INFO Request* and receiving a matching *A2S_INFO Reply*).

- Repeat the previous steps until the Steam master server has no more game server details to return.

Each *getservers* query indicates the IP <address:port> pair of the last CS:S game server probed by the client (or 0.0.0.0:0 if this is the initial query) and is repeated if no *getserversResponse* reply is received. In this way the process compensates for loss of a previous *getserversResponse* packet.

A2S_INFO Request UDP/IP packets are 53 bytes long and *A2S_INFO Reply* packets can average 135 bytes or longer. (Note: Valve have no specific names for the messages between master server and client, so *getservers*, and *getserversResponse* were chosen for clarity. Additional details of Steam packet formats are online at Valve Corporation (2008)).

Figure 1 A steam client discovering CS:S game servers (see online version for colours)



3.2 Wolfenstein Enemy Territory

ET was released in 2003 as a free, online-only team-play FPS game, and still has a modest online player community. It is based on the earlier Quake III Arena (Q3A) game engine (released by idSoftware in 1999), and inherited Q3A's relatively simple server discovery mechanism. ET server discovery uses similar steps to Steam's approach in Figure 1, with some small differences in message sequencing and naming.

Public ET game servers register themselves at *etmaster.idsoftware.com*, the ET master server. When a player initiates server discovery through their ET client's server browser an ET client will:

- Issue a *getservers* query packet to UDP port 27950 of *etmaster.idsoftware.com*.
- Receive one or more *getserversResponse* packets from the master server (identifying the <IP address:port> pairs of all registered ET game servers).

- Begin issuing *getinfo* probes to each game server, eliciting one *infoResponse* reply packet for each *getinfo* probe.
- Populate its on-screen server browser with details from each *infoResponse* reply (with an RTT estimated from the time between sending a *getinfo* and receiving a matching *infoResponse*).

A single *getserversResponse* is a UDP/IP packet up to 810 bytes long carrying IP address:port pairs for up to 112 game servers. Multiple *getserversResponse* packets are returned in quick succession if there are more than 112 game servers currently registered. Client *getinfo* probes are 43-byte UDP/IP packets and *infoResponse* packets may be hundreds of bytes long. ET's *getserversResponse* packets contain no sequence numbers, so clients cannot be sure they are aware of all available game servers. (Additional details of ET discovery packet formats are online at (Armitage, 2007a)).

ET inherited a notable weakness of Q3A's server discovery mechanism. Client Y can forge an ET *getservers* query packet having a different source IP address X, causing the burst of multiple *getserversResponse* reply packets to arrive at host X rather than Y. The multiplicative impact may be used to instantiate indirect denial of service attacks against innocent targets. FPS developers have learnt from this issue, so (for example) CS:S reduces the problem by requiring a separate *getservers* query packet for each *getserversResponse* reply. Inundating an innocent target with 10000 Steam *getserversResponse* packets would require a malicious host to send 10000 *getservers* packets to the Steam master server.

4 Background noise, 24 hours per day

The topological and/or geographic distribution of potential player populations may be estimated by monitoring the origins of probes to an individual game server located anywhere in the world (even if the game server itself is small and un-visited).

FPS server discovery results in a continual influx of probe packets to individual game servers, 24 hours a day, 7 days a week. Probes come in from thousands (or tens of thousands) of *potential* players and automated game server monitoring systems (such as ServerSpy, 2008). This 'background noise' of probe traffic fluctuates every 24 hours (following the waking and sleeping cycle of population centres around the world) and continues while ever a game server remains registered with the master server.

Figure 2 illustrates this phenomenon from the perspective of a CS:S game server running on port 27016 of PlanetLab (2008) host planetlab2.csg.uzh.ch (based in Switzerland) in early 2008. The number of unique IP addresses seen in a sliding 600-second window is plotted for probes coming from (respectively) the entire World, from Germany, and from the USA. Probes clearly originate from each country primarily when potential players are awake. CS:S game servers hosted elsewhere around the world experience similar 24-hour trends.

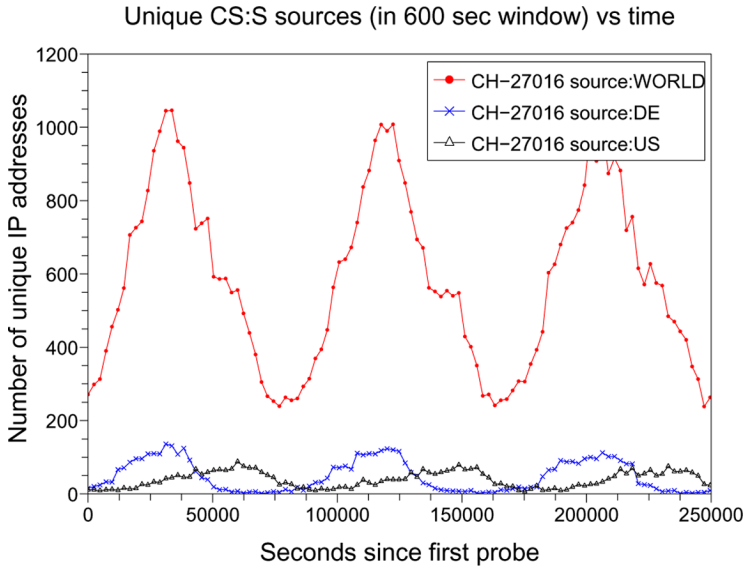
Player-initiated ET probe arrivals have been shown to be uncorrelated and exhibit exponentially distributed inter-probe intervals during both busiest and least-busy hours of their 24-hour cycle (Armitage, 2007b). (Unpublished companion work to Armitage (2007b) revealed CS:S probe arrivals exhibiting similar characteristics.

Player-triggered probing for other types of FPS games is likely to exhibit similar patterns.)

Probe traffic patterns do not necessarily correlate with actual game-play traffic patterns. A game server will be probed because it is registered with the master server. Only later will the game server be chosen (or rejected) for active game-play based on its attractiveness to potential players (such having acceptable RTT, good maps, or friends playing on a particular server) (Zander et al., 2005). Consequently, to track potential player populations is it sufficient to register a game server that supports no players and monitor the subsequent trends in probe arrivals.

A side effect is that if network devices that keep per-flow state (such as stateful firewalls or network measurement tools) are exposed to game server traffic they must be provisioned to cope with the number of UDP flows created by server discovery probing, regardless of any limits imposed on the number of concurrent players allowed on the game server. (Figure 2 shows that if per-flow state is kept for 10 min, resources for up to 1000 flows are required. Although not shown, this same game server peaked at around 5400 unique server discovery flows over 60 min periods, and 10 K unique flows over 120 min periods.)

Figure 2 Number of uniques probe sources seen every 600 seconds from Germany, USA and the entire World by a CS:S game server in Switzerland. The 24-hour cycles are influenced by people's waking and sleeping periods in each country (see online version for colours)



5 Not too slow and not too fast

Many FPS game clients reach the internet via dial-up or a home ethernet LAN connected to 'broadband' links in the 128 Kbps to 2 Mbps range (upstream). Probe too slowly and server discovery becomes a nuisance to the player. Send probes too fast and

transient congestion on the outbound link to the internet leads to over-estimation of RTT (or worse, complete loss of probe packets). Optimal probe transmission balances the player's needs for both timely *and* accurate server discovery.

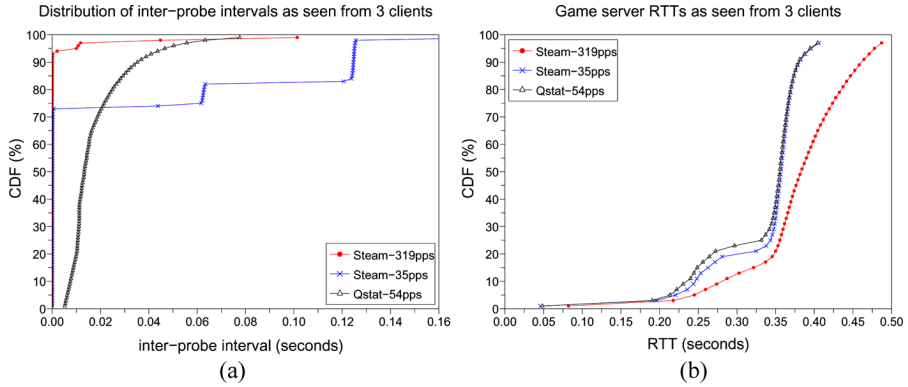
Figure 3 illustrates the issue when probing approximately 30,000 CS:S servers in February 2008 from Australia, using an open-source server browser (Qstat version 2.11 with default settings, capable of probing many different FPS game types QStat, 2008) and a standard Steam client (updated 9th January, 2008). The Steam client was configured for two different internet connection speeds - high speed ('DSL/Cable > 2M') and low speed ('modem – 56K'). In each case the client initated probes over a 100 Mbit/sec LAN to an ADSL2+ modem running 835 Kbit/sec upstream and 10 Mbit/sec downstream.

Figure 3(a) shows the distribution of intervals between consecutive *A2S_INFO Request* probes. Qstat's default behaviour averaged 54 probes/second, with a relatively broad range of inter-probe intervals. In contrast, the Steam client's high speed (319 probes/second) and low speed (35 probes/second) modes both emit bursts of *A2S_INFO Request* probes back-to-back (less than 1ms apart) inter-mingled with larger gaps in time.

Figure 3(b) shows the resulting RTT distributions. Each case reveals a small number of CS:S servers within Australia or Asia (~20 to ~140 ms), a moderate number of servers in North America (the ~200 ms+ range) and a far larger community of servers in Europe (~330 ms+). However, RTTs estimated by the Steam client in high speed mode are clearly skewed higher than those measured by Qstat or Steam client in low speed mode. In addition, even RTTs measured by the low speed Steam client show a small degree of upward skew relative to the Qstat measurements, despite averaging only 65% of Qstat's nominal probe rate. In both Steam cases the bursts of back-to-back *A2S_INFO Request* probes cause transient congestion in the ADSL2+ modem's uplink queues, adding extra delays to the probes occuring near the ends of each burst.

Figure 3's message is that clients who pace their emission of probe packets will achieve better RTT estimation accuracy than clients who merely cap the long-term average probe rate.

Figure 3 (a) Steam client inter-probe intervals are more bursty than Qstat and (b) steam RTT estimates are 'smeared' higher as the player's network link suffers transient congestion (see online version for colours)



6 Impact of master server rank

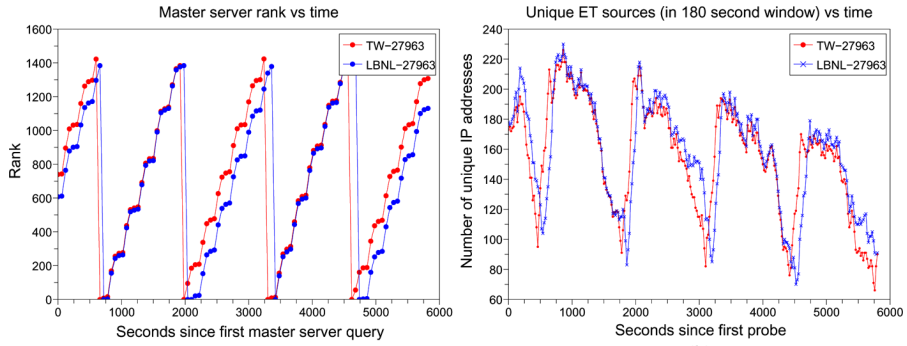
The level of probing experienced by your game server can be influenced by where your game server's IP address appears in the master server's list. This happens because server discovery takes time and humans are impatient. FPS clients typically allow players to terminate the server discovery process any time – whether to simply give up, or to immediately start playing on a server that is already been probed. Therefore servers towards the end of a master server's list are slightly less likely to be probed.

ET provides clear illustration of this issue. ET clients probe available game servers in the order they are listed in the master server's *getserversResponse* packets. The ET master server rotates the rank (distance from first place on the *getserversResponse* list) of every game server over periods of minutes. During each cycle a game server would find itself consecutively 1st, 2nd... *N*th in the list returned by the master server.

Figure 4(a) shows the rank (measured every 60 s) of two ET game servers vs. time in early 2008. (TW-27963 was an ET server on port 27963 of Planetlab host *planetlab2.iis.sinica.edu.tw*, whilst LBNL-27963 was on port 27963 of Planetlab host *node2.lbnl.nodes.planet-lab.org*. Both registered at much the same time, hence their similar ranks.) Figure 4(b) shows that the number of unique IP addresses seen probing these two game servers rises and falls in time with the game server's position on the master server's list. Figure 4(b)'s cyclic behaviour (repeating roughly every 22 min) is superimposed on a larger, 24-hour cycle due to the waking and sleeping times of ET players around the world (similar to the CS:S experience in Figure 2).

Server discovery mechanisms should try to rotate or randomise the rankings of individual game servers over time to ensure everyone has a fair chance of being probed by all potential players.

Figure 4 (a) ET master server cyclically alters the rank of two ET servers – one in Taiwan (TW-27963) and one in California (LBNL-27963) and (b) as each ET server moves further from rank one it sees less unique probes (in this case averaged over a 180-second interval) (see online version for colours)



(a) (b)

7 Probe sequence optimisations

A number of techniques exist or have been proposed to speed up the discovery of suitable game servers and reduce the number of probes emitted. Here we will describe

master-server filtering, client-side filtering, and probe re-ordering using CS:S as our illustrative example (although the basic ideas apply more generally). We will also touch on possible use of network coordinate systems.

7.1 Server- and client-side filters

In late 2007 and early 2008 the Steam master server has been returning around 28,000–33,000 CS:S servers, of which approximately 27,000–30,000 respond to probes. Steam provides server-side and client-side filters to manages this deluge of information.

Server-side filtering occurs when a player's initial *getservers* query requests game servers of a certain type (such as "only CS:S game servers" or "only Half-Life 2 Deathmatch servers") or servers believed (by the master server) to be in one of eight broad geographical regions of the planet (such as 'US-West', 'Europe', 'Asia', etc.). Filtering on geographical region means your client ends up probing only those game servers whose operators have self-identified as being in one of the eight Steam-defined regions.

Server-side filtering can reduce the number of game servers returned by the master server, reducing the subsequent number of *A2S_INFO Request/Reply* probes and the time spent probing.

Client-side filtering (such as not showing servers that are full or empty, or ranking the servers in order of ascending RTT) simplifies the server browser's presentation of information. However, it occurs during or after each active probe and has limited impact on the traffic generated during server discovery.

7.2 Probe re-ordering

Another approach is to modify FPS server discovery such that closer servers are probed and presented before more distant game servers. This can allow a player (or the client itself) to terminate probing once the RTT exceeds the player's personal threshold. However, this clearly involves an element of chick-and-egg – ranking game servers by probable RTT is hard to do before we have probed them to estimate their RTT.

Master servers cannot pre-sort their list of game servers in order of ascending RTT unless they know the network conditions likely to exist between any given client and every game server. In principle a master server might 'learn' from clients who have recently engaged in server discovery, and over time build up a map of historical RTTs between client locations and game server locations on the internet. This learned information might then be used to approximate the RTT (for example, by modifying a network coordinate system such as Dabek et al., 2004) that is likely to be experienced by a future client probing all the known game servers.

However, such an approach changes the trust model for enthusiast-hosted FPS games. The master server would now rely on clients providing realistic(and truthful) RTT estimates from different parts of the internet. Injecting large quantities of corrupt RTT information would be trivial under today's model of unauthenticated client access to the master servers. Consequently, the interface between clients and master server would have to evolve to include a more sophisticated trust and/or reputation model (and likely then involve tracking the identities of clients, in order to update each client's reputation as a provider of RTT estimates).

Another solution is to arrange game servers into *clusters* believed to have similar topological distance from the client, have the client probe a subset of the game servers

in each cluster, then use the resulting RTT estimates to re-order the subsequent probing of all remaining game servers. Clustering might be performed using countrycodes (Armitage, 2008a) or Autonomous System (AS) numbers (Armitage, 2008b).

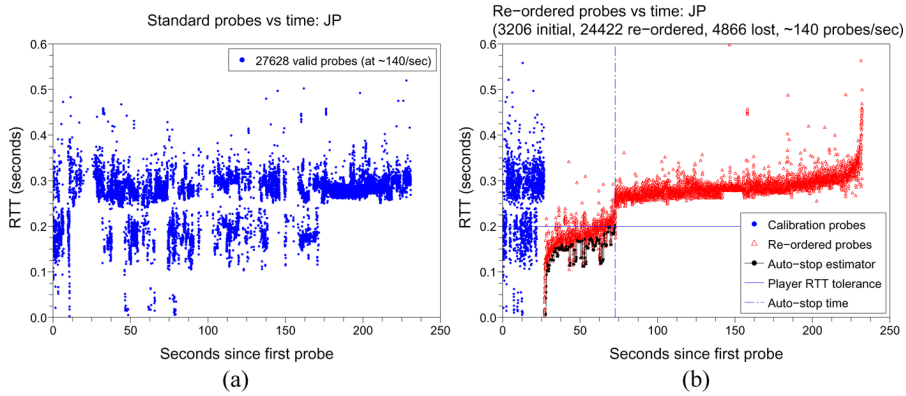
Figure 5 illustrates the use of AS numbers to cluster game servers before pre-probing a subset of the clustered game servers in order to rank the likely RTT of all remaining game servers. Figure 5(a) shows the individual RTT estimates seen by a regular Japanese CS:S client (plotted vs. time since the first server discovery probe). A player needs to wait for almost 230 s to be sure that they have found all the game servers who may have RTTs under 150–200 ms.

Figure 5(b) shows the re-ordered probe sequence with a nominal auto-stop threshold of 200 ms. During the first 25 s we *calibrate* the client by pre-probing a subset of game servers from each AS. This establishes a nominal RTT to each cluster. The clusters are then ordered by ascending RTT, and the client begins transmitted additional probes to all remaining game servers. After 73 seconds (31% of Figure 5(a)'s probe sequence) the auto-stop threshold is reached and the server discovery is complete.

Results for potential Australian, Taiwanese, UK, USA and German CS:S clients are also summarised in Armitage (2008b). For example, an Australian CS:S client's auto-stop may occur at only 15% of the normal 230 seconds, whereas a German CS:S client sees barely any improvement – auto-stop occurs at 97% of the normal 230 seconds. Clients who are close (under the player's RTT threshold) to most game servers (such as CS:S clients in Europe) see limited benefit. Fortunately, this scheme's worst case behaviour is no worse than normal CS:S server discovery.

Ultimately this scheme is beneficial any time there are thousands of game servers beyond a typical player's RTT tolerance threshold, even if the distribution of game servers happens to be relatively uniform around the planet. However, it requires modifications to clients and the master server. Armitage (2008b) suggests clustering game servers by their origin AS numbers at the master server, and returning pre-clustered sets of game servers to the client. The client would still perform the calibration pre-probing phase, but be freed from needing to maintain up to date routing table information.

Figure 5 (a) A Counter-Strike: Source client in Japan spends 230 seconds probing a set of game servers who are mostly too far away for competitive play and (b) re-ordering the probe sequence results in less probe packets to distant game servers and earlier termination of the server discovery search phase (see online version for colours)



8 Conclusions

With enthusiasts hosting thousands of individual game servers around the internet, FPS game developers continue to refine relatively simple schemes so that player-initiated server discovery can be prompt, accurate and use minimal network resources. However, the growing popularity of FPS games leads to situations where, for example, over 30,000 Counter-Strike: Source must be probed sequentially each time a player initiates server discovery. Future evolution of server discovery will almost certainly need to build in proactive information sharing between clients and master servers to optimise their search sequences, particularly with a view to eliminating searches of game servers known (or strongly believed) to be beyond the player's RTT tolerance. We have outlined one recently published approach that infers topological locality between different game servers (e.g., by common country code or AS number), estimates the RTT to a small subset of servers in each locality, then uses these RTT estimates to re-order the probing of all remaining game servers. Depending on a client's location, this approach can cut server discovery time and traffic by over 80%.

References

Armitage, G. (2007a) *Server Discovery for Quake III Arena, Wolfenstein Enemy Territory and Quake 4*, CAIA Technical Report 070730A, http://caia.swin.edu.au/reports/070730A/CAIA-TR-070730A.pdf, July.

Armitage, G. (2007b) 'A packet arrival model for wolfenstein enemy territory online server discovery traffic', *Proceedings of 15th IEEE International Conference on Networks (ICON)*, Adelaide, Australia, November, pp.31–36.

Armitage, G. (2008a) 'Client-side adaptive search optimisation for online game server discovery', *Proceedings of IFIP/TC6 Networking 2008*, Singapore, May, pp.494–505.

Armitage, G. (2008b) 'Optimising online FPS game server discovery through clustering servers by origin autonomous system', *Proceedings of ACM NOSSDAV 2008*, Braunschweig, Germany, May, pp.3–8.

Armitage, G., Claypool, M. and Branch, P. (2006) *Networking and Online Games – Understanding and Engineering Multiplayer Internet Games*, John Wiley & Sons, Ltd., UK, June.

Chambers, C., Feng, W-C. and Saha, D. (2003) 'A geographic, redirection service for on-line games', *ACM Multimedia 2003*, Short Paper, November.

Dabek, F., Cox, R., Kaashoek, F. and Morris, R. (2004) 'Vivaldi: a decentralized network coordinate system', Vol. 34, No. 4, ACM, New York, USA, pp.15–26.

Feng, W-C. and Feng, W-C. (2003) 'On the geographic distribution of on-line game servers and players', *NetGames '03: Proceedings of the 2nd Workshop on Network and System Support for Games*, ACM Press, New York, USA, pp.173–179.

Henderson, T. (2002) 'Observations on game server discovery mechanisms', *NetGames '02: Proceedings of the 1st Workshop on Network and System Support for Games*, ACM, New York, USA, pp.47–52.

Lee, Y., Agarwal, S., Butcher, C. and Padhye, J. (2008) 'Measurement and estimation of network QoS among peer Xbox 360 game players', *Proc. 9th Passive and Active Network Measurement Conference (PAM 2008)*, Springer, Berlin/Heidelberg, April, pp.41–50.

PlanetLab (2008) *PlanetLab – An Open Platform for Developing, Deploying, and Accessing Planetary-Scale Services*, https://www.planet-lab.org/, accessed 8th February.

QStat (2008) http://www.qstat.org/, accessed 8th February.

ServerSpy (2008) *ServerSpy – Online PC Gaming Statistics*, http://www.serverspy.net/, as of 30th July.

Valve Corporation (2008) *Server Queries*, http://developer.valvesoftware.com/wiki/Server_Queries, as of 7th February.

Zander, S., Kennedy, D. and Armitage, G. (2005) 'Dissecting server-discovery traffic patterns generated by multiplayer first person shooter games', *Proceedings of ACM Networks and System Support for Games (NetGames) Workshop*, New York, USA, October, pp.1–12.