Client-Side Adaptive Search Optimisation for Online Game Server Discovery

Grenville Armitage

Centre for Advanced Internet Architectures
Swinburne University of Technology, Melbourne, Australia
garmitage@swin.edu.au

Abstract. This paper describes a client-side, adaptive search technique to reduce both the time taken to discover playable online First Person Shooter (FPS) game servers and the number of network flows created during game server discovery. Online FPS games usually use a client-server model, with thousands of game servers active at any time. Traditional FPS server discovery probes all available servers over multiple minutes in no particular order, creating thousands of short-lived UDP flows. Probing triggers rapid consumption of longer-lived per-flow state memory in NAT devices between a client and the internet. Using server discovery data from Valve's Counterstrike:Source and idSoftware's Wolfenstein Enemy Territory this paper demonstrates that pre-probing a subset of game servers can be used to re-order and optimise the overall probe sequence. Game servers are now probed in approximately ascending latency, expediting the location of playable servers. Discovery of playable servers may now take less than 20% of the time and network traffic of conventional game server discovery. The worst case converges to (without exceeding) the behaviour of conventional game server discovery.

Keywords: Server discovery, search optimisation, latency estimation.

1 Introduction

Internet-based multiplayer First Person Shooter (FPS) games (such as Wolfenstein Enemy Territory [1], Half-Life 2 [2], Counterstrike:Source [3] and Enemy Territory Quake Wars [4]) have become quite common in the past 6+ years. 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 [5]. The challenge for game clients is to locate up-to-date information about the game servers available at any given time, such that the player can select a suitable server on which to play.

Server discovery is usually triggered manually by the human player, to populate or refresh the list of available servers presented by their client's on-screen

A. Das et al. (Eds.): NETWORKING 2008, LNCS 4982, pp. 494–505, 2008.

'server browser'. Once triggered, a game client first queries a master server preconfigured into the game client software. The master server returns a list of thousands of <IP address:port> pairs numbers representing currently 'active' game servers. The client then steps through this list, probing each listed game server for information (such as the current map type, game type and number of players). The probe, a brief two-way UDP packet exchange, allows the client to estimate the current latency (round trip time, RTT) between itself and each game server who replies. All this information is presented to the player (usually as it is gathered), who then selects a game server to join.

A key server selection criteria is the client-to-server latency. Published literature suggests that competitive online FPS game play requires latencies below roughly 150ms to 200ms [5]. However, from many locations on the Internet there will be game servers over 200ms away. Players cannot know which servers fall under their latency tolerance until the server discovery process has completed. A given client will send out hundreds or thousands of probe packets before the player selects and joins only one game server.

There are two noteworthy client-side consequences. First, server discovery can take multiple minutes to probe all available game servers. Second, any network address translation (NAT) devices between the client and the internet will experience a burst of dynamically-created state entries in its NAT tables (potentially creating temporary exhaustion of the NAT device's table space).

This paper describes a self-calibrating client-side method for optimising the server discovery probe sequence. The method adapts to wherever the client is located (topologically) on the Internet, and does not require the client to have a priori knowledge of its public IP address. The time to discover playable servers can be 20% or less of the regular server discovery time (and in the worst case, converge on - without exceeding - the time required for regular server discovery). The method is illustrated with examples based on Valve's Counterstrike:Source (CS:S) and idSoftware's Wolfenstein Enemy Territory (ET).

The rest of this paper is organised as follows. Section 2 provides more details of existing server discovery techniques and their limitations. This paper's proposed method is described in section 3, with section 4 illustrating the proposed technique's potential impact. Limitations, alternatives and future work are outlined in section 5. The paper concludes in Section 6.

2 Current State of FPS Server Discovery

In this section we review the current ET and CS:S server discovery processes, consider how they impact on a player's experience, and reflect on the network layer constraints and consequences for consumer broadband services.

2.1 ET and CS:S Server Discovery

Figure 1 illustrates the server discovery process used by ET. Released in 2003 as an online-only team-play FPS game, ET is based on the earlier Quake III

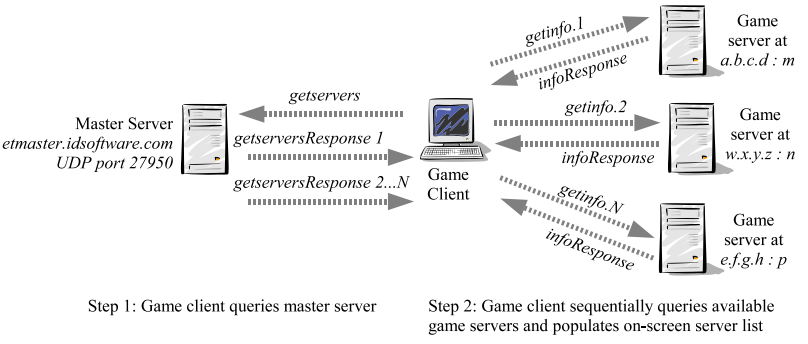

Fig. 1. An Enemy Territory (ET) client's discovery and probing of registered ET game servers

Arena (Q3A) game engine, and inherits Q3A's underlying server discovery mechanism. Public ET game servers automatically register themselves at etmaster.idsoftware.com, the ET *master server*.

When a player requests server discovery an ET client first queries the ET master server (*getservers*), retrieving a list of <IP address:port> pairs. This list (returned in one or more *getserversResponse* packets) represents all currently registered game servers. The client then probes each server (using 43-byte *getinfo* UDP/IP packets) in the order provided by the master server, eliciting *infoResponse* replies from each game server. The client's on-screen 'server browser' is populated with game server information (and estimated RTT) as each *infoResponse* reply returns.

CS:S has been regularly updated since first released in late 2004 by Valve Corporation. Server discovery uses Valve's Steam [6] online authentication and game delivery system. Although the packet syntax [7] differs markedly to the protocol used by ET the basic steps are similar to those shown in Figure 1. Players initiate server discovery through their Steam client's game server browser. The client retrieves a list of <IP address:port> pairs from a Steam master server at hl2master.steampowered.com, and begins probing game servers with 53-byte UDP/IP packets containing the ASCII string *TSource Engine Query* (TSEQ). The Steam client's server browser is updated as replies comes back.

2.2 Real World Examples

For this paper representative data was gathered from a number of locations around the planet (listed in Table 1). The Australian host ran actual ET and CS:S clients, whilst the others were nodes on Planetlab [8] using the open-source game server discovery tool qstat [9].

Figures 2 and 3 illustrate the distribution of game server RTTs versus probe time experienced by the standard ET and CS:S clients in Australia in late September 2007. For ET the distribution of RTTs is consistent (and widely spread) across the 70 second discovery period. For CS:S the distribution is not

Table 1. Location of clients used to gather representative RTT samples

Host name and location	Country and client type(s)
gjagw.space4me.com, Australia (actual clients)	AU-CSS, AU-ET
planetlab1.otemachi.wide.ad.jp, Japan (Planetlab)	JP-ET, JP-CSS
planetlab2.csg.uzh.ch, Switzerland (Planetlab)	CH-ET
edi.tkn.tu-berlin.de, Germany (Planetlab)	DE-CSS

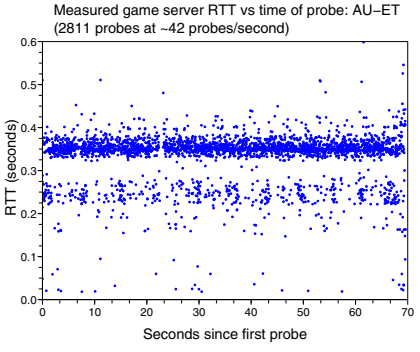

Fig. 2. Measured RTT vs time: ET server discovery sequence as seen from Australia in September 2007

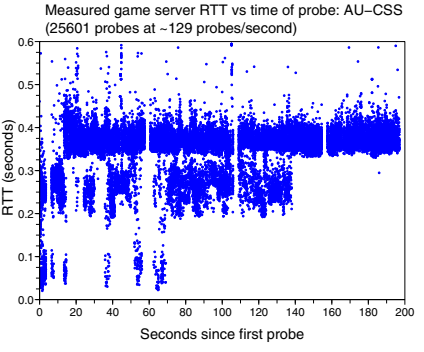

Fig. 3. Measured RTT vs time: CS:S server discovery sequence as seen from Australia in September 2007

as uniformly spread across the (roughly) 3 minute discovery period. In both cases all servers must be probed before players can presume they've covered all possible game servers with 'playable' RTT.

The large number of probes over 200ms in Figures 2 and 3 reflects the fact that many ET and CS:S game servers tend to be hosted in Europe and the USA. Similar RTT levels are seen from Japan, whilst the clients in Germany and Switzerland saw almost the inverse - most probes returned low RTTs, under 200ms. ET and CS:S clients in the Asia-pacific region end up probing many servers that are subsequently determined to be unsuitable for competitive play.

2.3 Impact on Consumer Network Connections

Most game clients sit behind consumer broadband connections. The server discovery probe rate is consequently capped to the client's estimate of available upstream bandwidth. FPS clients usually require players to indicate their Internet connection type ('dial-up', 'ADSL', 'Cable modem', etc). In addition to tweaking the packet rates used during actual game play, this information can allow the client to estimate an appropriate server discovery probe rate. Too many probes per second may congest the player's upstream bandwidth, potentially inflating the estimated RTT or causing probe packets to be dropped. Too few probes per second increases the time taken to complete server discovery.

Server discovery also increases memory consumption in NAT-enabled routers between the client and the Internet. Although UDP is notionally state-less, NAT devices typically retain mapping state for a number of minutes after seeing UDP traffic head to the Internet from a local network. For example, the trial for Figure 3 caused ~25K unique NAT mapping table entries to be dynamically created over a 3 minute period. Despite each server discovery transaction completing in under two seconds these NAT table entries remained (wasting memory) for multiple minutes before being released.

2.4 Filtering at the Client and Master Server

Client-side filtering (such as not showing servers that are full or empty, or ranking the servers in order of ascending RTT) has limited impact on the network traffic created during server discovery because it happens during or after the active probing of the available game servers. (Players may certainly choose to exit server discovery early when they see enough servers with tolerable. Nevertheless, as Figures 2 and 3 show, the player cannot safely assume they're seeing all servers under a particular RTT until all available servers have been probed.)

Valve's Steam master server supports rudimentary server-side filtering. Manually selected by the player, a Steam client's initial query can request game servers of a certain type (such as "only CS:S game servers") or game 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). Filters reduce the number of game servers returned by the master server, and hence reduce the number of probes subsequently emitted by a Steam client.

However, a master server cannot know a priori the RTT between any given client and game server. Thus a client must still actively probe the entire (albeit possibly reduced) set of IP address:port pairs handed back by the master server.

3 Proposed Client-Side Optimisation

This paper's client-side optimisation meets four key goals:

- Presentation of results from closer servers before those of more distant servers
- Optional automatic early termination of search sequence
- Function from behind consumer NAT devices without additional manual intervention or configuration by the player
- No additional processing load on master servers

The first two goals improve a player's experience and reduce network traffic generated by server discovery. The third goal addresses usability. Players should not be expected to manually configure additional knowledge into the client (such as the public IP address of the player's home broadband connection). Adapting to being behind a NAT device is crucial because so many game clients will sit on a consumer home broadband connection. The fourth goal minimises the incremental operational cost to a game publisher (as master servers already serve thousands of queries per hour without generating any new revenue).

3.1 Background

The challenge of finding FPS servers with low enough RTT is well recognised [10]. To date research has focused on re-locating clients to optimally placed servers (e.g. [11]), rather than optimising the server discovery process itself.

The problem appears contradictory: we wish to probe game servers in order of ascending RTT before we've probed them to establish their RTT. In 2006 the author hypothesised that a client might locally re-order the probe sequence so that game servers in countries 'closer' to the client would be probed before those 'further away' [12]. First the client would map server IP addresses to their country of origin (for example, using MaxMind's free GeoLite Country database [13]). Then selected servers in each country would be probed, providing an estimate of the RTT to each country relative to the client's current location. Finally, the countries would be ranked in ascending order of estimated RTT, and all remaining game servers probed in order of their country's rank. Subsequent implementation revealed that re-ordering solely on the basis of country code was inadequate. This paper presents a functional improvement of the idea in [12].

3.2 Re-Ordering Based on Country and Topology

Algorithm 1 shows the proposed steps - clustering, calibration and re-ordered probing. Clustering involves identifying those game servers who share a *common topological relationship within a country*. Calibration involves initially probing a small subset of servers in each cluster, thereby creating an estimate of the RTT to every server in the cluster. (The cluster is sub-divided if the sampled RTTs are spread across 'too wide' a range.) Re-ordered probing involves ranking clusters by ascending RTT then probing remaining game servers according to their cluster's rank.

The definition of a cluster relies on two facts. First, third-party databases allow geographical location to be inferred from IP addresses, even though an IP address by itself has no geographic semantics. Second, within the context of given country the higher order IP address bits provide a coarse, but sufficient, key to differentiate servers sharing paths with potentially different latencies.

Calibration takes care of two key requirements. Active sampling ensures cluster RTTs for ranking are relative to the client's current Internet location (whether or not the client sits behind a NAT device). In addition, a cluster covering too wide a spread of RTT values is subdivided along /16 boundaries, creating multiple smaller clusters for the ranking step. The game servers probed in step 4 are considered 'done' and not probed again in step 6.

Modern FPS games typically require processor speeds well over 1GHz, so Algorithm 1 adds negligible overhead relative to the time required to actually transmit and receive all probes. Thus in the worst case this algorithm's performance (timeliness in presenting acceptable servers to the player) converges on, without being worse than, the performance of conventional server discovery.

Algorithm 1. Clustering, calibration and re-ordered probing

1. Query master server for list of game servers
2. Create initial clusters:
   (a) Assign every game server a country code
   (b) Identify every /8 subnet within each country code that contains:
      i. more than 10 game servers, and
      ii. more than 10% of all game servers from that country
   (c) Every server in a /8 subnet identified above is assigned a unique *cluster_id* from the server's country code and the /8 subnet number
   (d) Every server not in a /8 subnet identified above is assigned the server's country code as a unique *cluster_id*
3. Servers with the same *cluster_id* are now members of the same cluster
4. For each cluster_id, perform the following steps:
   (a) Set $N_{sample} = \sqrt{N_{cluster}}$, where $N_{cluster}$ is the number of servers in the cluster
   (b) Randomly select N_{sample} servers from the cluster, one from each /16 subnet present within the cluster
   (c) Issue a standard server discovery probe to each of the server's selected in the previous step
   (d) The estimated RTT ($RTT_{cluster}$) for this cluster is the median measured RTT
   (e) If the 20th and 80th percentile measurements making up $RTT_{cluster}$ differ by more than 40ms:
      i. Split the members of this cluster_id into mutiple new clusters
      ii. Each new cluster (and associated cluster_id) is made from members of the old cluster who share the most significant 16 bits of their IP addresses
      iii. Issue a standard server discovery probe to one previously un-probed server randomly selected from each new cluster. The probed RTT becomes $RTT_{cluster}$ for the new cluster
5. Rank every cluster in order of ascending $RTT_{cluster}$
6. Probe all remaining game servers in order of their cluster's rank. Within a cluster, probe servers in the order they were returned by the master server.

3.3 Automatic Termination of Discovery Phase

Individual servers probed in Algorithm 1's step 6 may not have ascending RTTs, but the RTT averaged over a number of recently probed servers will trend upwards. Consequently it becomes feasible to implement automatic early termination (auto-stop) of probing during Step 6 when the RTT trends above a player-specified threshold. This will reduce both the number of probes transmitted and the time a player must wait to be reasonably sure the client has probed the servers who are acceptably close (particularly for clients a long way from many servers). The variability of individual RTTs for game servers within a given cluster mean an auto-stop decision should err on the side of continuing rather than terminating the search. Algorithm 2 has been found to be suitably cautious.

Algorithm 2. Auto-stop calculation

- Set RTT_{stop} as the maximum RTT considered playable for a game server (for example, $\text{RTT}_{stop} = 200\text{ms}$)
- Decide on a window size $\text{W}_{autostop}$ (for example, $\text{W}_{autostop} = 100$)
- Wait until at least $\text{W}_{autostop}$ servers are probed in Algorithm 1's step 6.
- Now, after each successive probe in Algorithm 1's step 6 re-calculate RTT_{bottom} as the 2nd-percentile RTT over the last $\text{W}_{autostop}$ probes (the RTT below which 2% of RTT samples have fallen)
- Terminate Algorithm 1's step 6 when $\text{RTT}_{bottom} \geqslant \text{RTT}_{stop}$

4 Illustrating the Optimisation with CS:S and ET

Space constraints preclude an exhaustive analysis of section 3's client-side optimisation. Instead, we present a selection of illustrative examples using real-world RTT datasets gathered from the hosts in Table 1. Each dataset was then used to simulate the consequences of a client applying the steps in sections 3.2 and 3.3. Quantitative results are summarised in Table 2. (Differences exist between the number of servers probed in each case as the datasets were not gathered at precisely the same time.)

For each simulated client Algorithm 1 generates traffic in two phases - 'initial probes' (step 4) and 're-ordered probes' (step 6). Only re-ordered probes are shown on each graph, along with the variation of Algorithm 2's RTT_{bottom} over time. The auto-stop RTT threshold is 200ms, and the estimated auto-stop time is indicated by a blue vertical line.

Figures 4, 5 and 6 illustrate the impact on typical ET clients located in Australia, Japan and Switzerland respectively. Compared to the ~70 second probe sequence in Figure 2 the Australian client sees a distinct improvement. It takes ~10 seconds to cluster the active game servers and begin issuing re-ordered probes, with auto-stop being triggered in ~14 seconds. A similar benefit is evident for the client based in Japan. Figure 6 illustrates the neutral impact on clients close to large concentrations of game servers under 200ms. Servers over 200ms are seen towards the end of the probe sequence, but the auto-stop process errs on the side of continuing (rather than stopping) and the client ultimately probes all active servers.

Table 2. Impact of utilising optimised server discovery from different locations

Client	Countries	/8 clusters	Initial Probes	Auto-stop time (seconds)	Auto-stop (% of full probe)
AU-ET	51	77	396	14	20
JP-ET	52	77	446	24	36
CH-ET	50	79	492	68	100
AU-CSS	71	140	2136	24	12
JP-CSS	69	143	1694	60	28
DE-CSS	69	143	2159	208	98

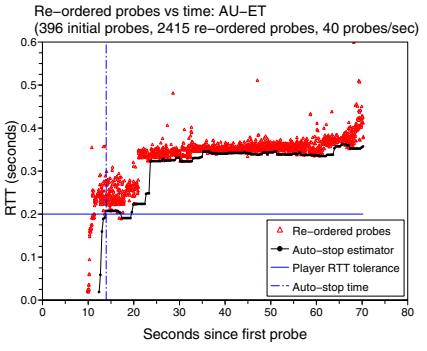

Fig. 4. Re-ordered ET probe sequence for a client in Australia

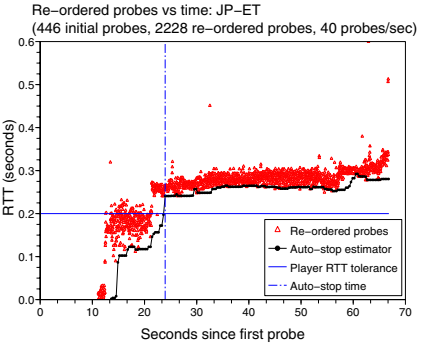

Fig. 5. Re-ordered ET probe sequence for a client in Japan

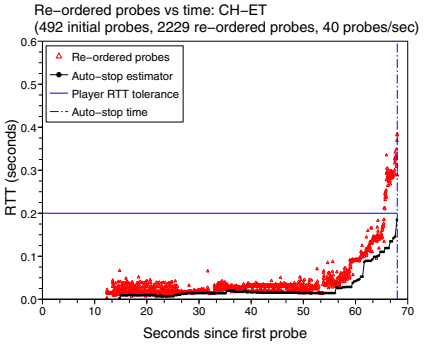

Fig. 6. Re-ordered ET probe sequence for a client in Switzerland

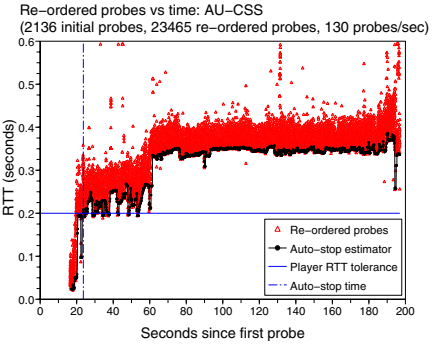

Fig. 7. Re-ordered CS:S probe sequence for a client in Australia

Figures 7, 8 and 9 illustrate the impact on typical CS:S clients located in Australia, Japan, and Germany respectively. Compared to the ~200 second probe sequence in Figure 3 the Australian client sees a distinct improvement. It takes just under 20 seconds to cluster the active game serves and begin issuing re-ordered probes, with auto-stop being triggered in ~24 seconds (12% of the non-optimised search time). A similar, albeit less dramatic, benefit is evident for the client based in Japan. A client based in Germany would see only limited benefit from the re-ordered probe sequence - the auto-stop triggers after almost 98% of active servers have been probed.

The key message is that clients far away from most game servers can see significant reduction in the number of probes emitted by their clients before concluding that all playable servers have been seen. This reduces the player's wait time, reduces the number of UDP flows passing through any NAT devices near the client, and reduces the number of bytes sent and received to probe servers unlikely to be suited to competitive play.

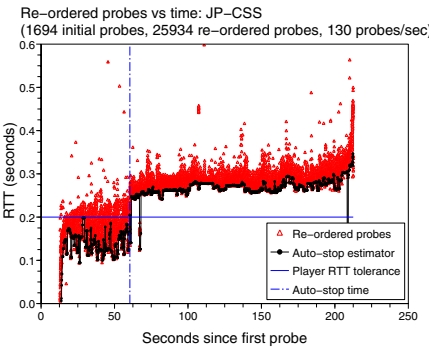

Fig. 8. Re-ordered CS:S probe sequence for a client in Japan

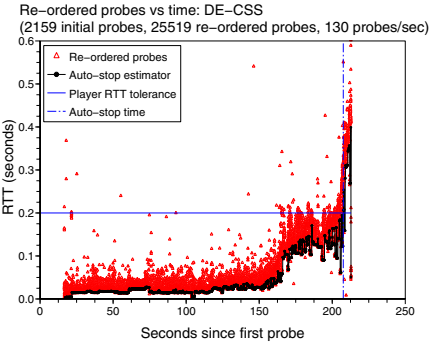

Fig. 9. Re-ordered CS:S probe sequence for a client in Germany

5 Limitations, Alternatives and Future Work

5.1 Limitations and Alternatives

It is inherently challenging to consistently optimise the full search sequence based on sampling of the master server's list of active game servers.

For example, Algorithm 1's N_{sample} should be kept low so the initial probe sequence in step 3 involves a small fraction of the game servers returned in step 1. This paper's calculation of N_{sample} works reasonably well for the datasets used. However, increasing N_{sample} will improve the estimation of $RTT_{cluster}$ and the reliable sub-division of clusters where necessary, improving the consistency of cluster ranking in step 6.

The current scheme relies on a third-party database for mapping IP addresses to geographically significant country codes. Such databases are themselves not perfect (MaxMind's GeoIP claims to map 97% of all IP address allocations to country codes), and they must be updated regularly as real-world IP address allocations change. Fortunately, the databases are small (relative to today's game clients) and don't change much from month to month (GeoLite Country went from 677Kbyte to ˜1Mbyte between April and October 2007). Most FPS games include mechanisms for auto-update of game content, which can also handle incremental updates to the country code mapping database.

Another issue is that auto-stop relies on an inherently noisy 'signal', and may be fooled into premature termination if the cluster ranking is sufficiently mis-ordered or RTT_{stop} is set too low.

Further options open up if we allow modification of the master server. Mapping of IP addresses to country codes could be moved to the master server, with tuples of <ipaddr:port:countrycode> returned to querying clients. Clients would skip step 2(a) of cluster formation, and discard the local address mapping database. However, reply packets from the master server would increase by two bytes per game server (assuming two-character country codes).

In principle a master server could also pre-order the list of game servers returned to each client by estimating the client's distance to servers in other countries (using the client's public IP address and a country code mapping database held at the master server). However, a master server cannot know the network conditions prevailing between each client and game servers in different countries, so there is no benefit to be gained.

5.2 Future Work

Future work will characterise the probability with which clusters are formed and ranked suboptimally as a function of N_{sample} , the use of /8 and /16 boundaries, and the spread of RTTs used to trigger new cluster formation in step 4(e). We will also evaluate alternative auto-stop algorithms, such as waiting for a *smooth* run of consistently high probe RTTs before stopping (rather than simply triggering on the lowest 2nd percentile RTT).

We will also explore alternative indicators that game servers share a common 'distance' from a client, such as Autonomous System (AS) numbers. AS numbers are used in inter-domain routing to identify topologically distinct regions of the internet. Clustering based on AS numbers (rather than country code) may lead to more accurate $RTT_{cluster}$ estimates and greater consistency in ranking of clusters. Master servers are a good place to track the live BGP routing information updates required to maintain IP address to AS number mappings. We will explore the implications of modifying master servers to return <ipaddr:port:ASnumber> tuples to querying clients.

6 Conclusion

Using examples based on Valve's Counterstrike:Source (CS:S) and idSoftware's Wolfenstein Enemy Territory (ET) this paper illustrates a self-calibrating client-side method for optimising the FPS game server discovery probe sequence. Game servers are clustered by their countries of origin, and the clusters are then ranked in ascending order of RTT based on initial probing of a small sample of game servers in each cluster. All remaining game servers are then probed in order of their cluster's rank. Probing may be automatically terminated when RTT exceeds a player-specified threshold. The method works wherever the client is located on the Internet, and can reduce server discovery time and traffic down to less than 20% of the regular case. In the worst case, the method converge on - without exceeding - the time and resources required for regular server discovery.

Acknowledgement

I am grateful to Mark Claypool for providing me with access to PlanetLab.

References

1. id Software: Wolfenstein Enemy Territory, under Downloads (September 29th 2007), http://www.enemyterritory.com/main.html
2. Valve Corporation: Half-Life 2 (April 29th 2007), http://half-life2.com/
3. Valve Corporation: CounterStrike: Source (accessed, February 8th 2008), http://counter-strike.net/
4. id Software: Enemy Territory Quake Wars (September 29th 2007), http://www.enemyterritory.com/
5. Armitage, G., Claypool, M., Branch, P.: Networking and Online Games - Understanding and Engineering Multiplayer Internet Games, June 2006. John Wiley & Sons, Ltd, United Kingdom (2006)
6. Valve Corporation: Welcome to Steam (September 27th 2007), http://www.steampowered.com/
7. Valve Corporation: Server Queries (February 7th 2008), http://developer.valvesoftware.com/wiki/Server_Queries
8. PlanetLab: PlanetLab - An open platform for developing, deploying, and accessing planetary-scale services (accessed, February 8th 2008), https://www.planet-lab.org/
9. : QStat (accessed, February 8th 2008), http://www.qstat.org/
10. Claypool, M.: Network characteristics for server selection in online games. In: ACM/SPIE Multimedia Computing and Networking (MMCN) (January 2008)
11. Chambers, C., Feng, W.C., Saha, D.: A geographic, redirection service for on-line games. In: ACM Multimedia 2003 (short paper). (November 2003)
12. Armitage, G., Javier, C., Zander, S.: Topological optimisation for online first person shooter game server discovery. In: Proceedings of Australian Telecommunications and Network Application Conference (ATNAC) (December 2006)
13. MaxMind: GeoLite Country (accessed, February 8th (2008), http://www.maxmind.com/app/geoip_country