Optimising Online FPS Game Server Discovery through Clustering Servers by Origin Autonomous System

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

ABSTRACT

This paper describes the use of origin Autonomous System (AS) information to optimise online First Person Shooter (FPS) game server discovery. Online FPS games typically use a client-server model, with thousands of game servers active at any time. Traditional server discovery probes all available servers over multiple minutes in no particular order, creating thousands of short-lived UDP flows. Using Valve's Counterstrike:Source game this paper demonstrates a multi-step process: Sort available game servers by origin AS, probe a subset of servers in each AS, rank each AS in ascending order of estimated round trip time (RTT), then probe all remaining game servers according to the rank of their origin AS. Probing game servers in approximately ascending RTT expedites the identification of playable servers. This new approach may take less than 20% of the time and network traffic of conventional server discovery (without exceeding conventional server discovery time and traffic consumption in the worst case).

Categories and Subject Descriptors

C.2.4 [**Computer-Communication Networks**]: Distributed Systems—*client/server, Distributed applications*

General Terms

Measurement, Performance

Keywords

Server discovery, search optimisation, latency estimation

1. INTRODUCTION

Internet-based multiplayer First Person Shooter (FPS) games generally 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 time [1]. The challenge for game clients is to locate up-to-date information about active game servers so a player can select a suitable server on which to play.

Players trigger *server discovery* to populate or refresh their game client's on-screen 'server browser' (a list of available game servers). Clients first query a well-known master server, which returns a list of all registered game servers (usually broken over multiple reply packets, as the lists can be quite long). Clients then probe each game server in turn for information such as the current map type, game type, and round trip time (RTT). Using this information the player then selects a game server to join.

RTT is a key server selection criteria - published literature suggests that competitive online FPS game play requires latencies below 150ms to 200ms [1]. Many servers and clients are over 200ms apart yet players cannot know which servers are suitably close until each server is probed. A client will send out thousands of probe packets before joining only one game server. In addition, server discovery can generate mega-bytes of network traffic while taking multiple minutes to complete. Network devices that keep per-flow state (such as NAT-enabled home routers) also experience a burst of dynamically-created state entries, tying up memory for minutes simply to enable a sub-second packet exchange.

This paper describes an optimised server discovery probe sequence that works wherever the client is located on the Internet. The time to discover playable servers can be less than 20% of the regular server discovery time (whilst converging on, without exceeding, the regular server discovery time in the worst case). Knowledge of each game server's origin Autonomous System (AS) enables clustering of servers likely to exhibit similar RTTs, and allows a client to re-order the probing to hit low RTT servers before high RTT servers. The technique is illustrated using Valve Corporation's Counterstrike:Source (CS:S) [2], yet generalises to many other online FPS games.

The rest of this paper is organised as follows. Section 2 details current CS:S server discovery and its implications. The proposed optimisation is described in section 3, with section 4 illustrating the optimisation's potential impact. Limitations, alternatives and future work are outlined in section 5. The paper concludes in Section 6.

2. COUNTERSTRIKE: SOURCE

This section reviews the CS:S server discovery process and the resulting player experience.

NOSSDAV '08 Braunschweig, Germany

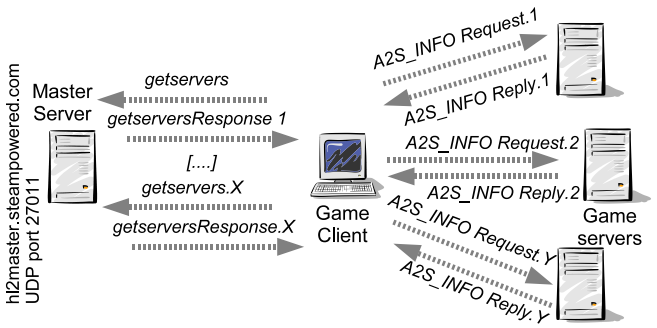

Figure 1: A Steam client discovering CS:S game servers

2.1 Valve's CS:S Server Discovery process

CS:S was first released in late 2004 by Valve Corporation through their Steam online game delivery system. Public CS:S game servers register themselves with Steam's *master server* at hl2master.steampowered.com. Players initiate server discovery through their Steam client's game *server browser*. Figure 1 illustrates the key server discovery steps. (Note, Valve have no specific names for the messages between master server and client, so *getservers*, and *getserversResponse* have been chosen for clarity. UDP/IP packet formats for server discovery are available online [3].) As of late 2007, a Steam client would:

- Issue a *getservers* query 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 game server in order, eliciting an *A2S_INFO Reply* packet from every active game server
- Repeat the previous steps until the Steam master server has no more game server details to return

A game server's RTT is the time between the client sending an *A2S_INFO Request* and receiving the *A2S_INFO Reply*. Server-specific information in each *A2S_INFO Reply* is used to update the Steam client's on-screen server browser (along with estimated RTT) as replies come back. This process allows players to discover servers on which they might wish to play.

The speed of server discovery is limited by a player's network connection. Too many probes per second can congest the player's link, inflating individual RTT estimates or causing probe packets to be dropped. Players may configure their Steam client to assume a network connection of 'Modem - 56K', 'DSL > 256K', etc, thus influencing the *A2S_INFO Requests* transmission rate.

Other server browsers may use a different sequence. Qstat [4] (an open-source server browser) retrieves all registered servers first (using back to back *getserver* queries) before issuing *A2S_INFO Request* probes. Ultimately the same result - RTT to all active servers is estimated by probing them in the order their <IP address:port> pairs are returned by the master server.

2.2 Client and master server filtering

In recent months the Steam master server returns around 28-31K CS:S servers, of which ˜27K respond to probes. Server-side and client-side filtering assists with 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 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). 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. It also reduces the amount of per-flow state created in network devices such as NAT-enabled home routers (which often retain new UDP flow mapping state for minutes after each sub-second *A2S_INFO Request/Reply* transaction).

Client-side filtering (not showing full or empty servers, or ranking 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.

2.3 Examples from around the planet

Examples of CS:S server discovery traffic from six different locations around the planet (Table 1) illustrate the potential for optimisation. Each location utilised qstat [4] to probe all available CS:S servers, capturing the UDP/IP traffic with tcpdump (to track actual network usage). 'Replies' indicates how many game servers actually sent back *A2S_INFO Replies*, 'Total Bytes' indicates how many IP-layer bytes were consumed performing the server discovery process, and 'Lost' indicates how many *A2S_INFO Requests* went unanswered (including repeats - qstat probes unresponsive game servers up to 3 times). The AU client was a home machine connected via ADSL2+, the rest were Planetlab nodes [5].

Table 1 reveals that ˜5Mbytes of network traffic is created during server discovery, regardless of a client's location. Outbound *A2S_INFO Request*s (53-byte UDP/IP packets) account for about 30%, with the remainder (inbound) made up of variable-length *A2S_INFO Replies*.

Figures 2 and 3 show the distribution of CS:S game server RTTs versus probe time experienced by the UK and Taiwan clients. At a nominal rate of 140 probes per second (approximating a Steam client configured for 'DSL > 256K' network access) server discovery takes ˜230 seconds to complete. (To save space, similar plots for AU, DE, USA and JP are not shown.) Regardless of a client's location on the planet, game server RTTs fluctuate right across the discovery period. Thus a player must wait for all game servers to be probed before they can presume to have seen all those with 'playable' RTT.

Figure 4 contrasts the RTT distributions experienced by all six clients. CS:S has many game servers in Europe, some in the USA and few in Asia. Game servers with RTT under 200ms are scarce for clients in AU, TW and JP, and common for clients in DE, UK and USA. Clients in the Asia-pacific region end up probing many game servers that are, realistically, unsuitable for competitive play.

3. PROPOSED OPTIMISATION

This paper's client-side optimisation to CS:S server discovery meets three key goals:

Client location	Replies	Total Bytes	Lost	Date
AU: Australia (author's home ISP connection)	30110	5787817	3414	Feb '08
DE: Germany, edi.tkn.tu-berlin.de	27678	5423561	5233	Nov '07
TW: Taiwan, planetlab1.iis.sinica.edu.tw	27673	5417405	5081	Nov '07
JP: Japan, planetlab1.otemachi.wide.ad.jp	27628	5397924	4866	Nov '07
UK: United Kingdom, planetlab4.cs.st-andrews.ac.uk	27638	5403846	4968	Nov '07
USA: United States, node2.lbnl.nodes.planet-lab.org	27639	5397293	4836	Nov '07

Table 1: CS:S client locations used to gather representative RTT samples

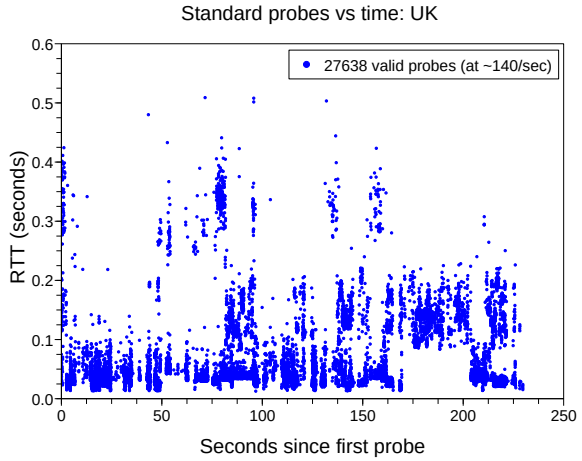

Figure 2: CS:S game server RTTs vs time as seen from the United Kingdom in Nov'07

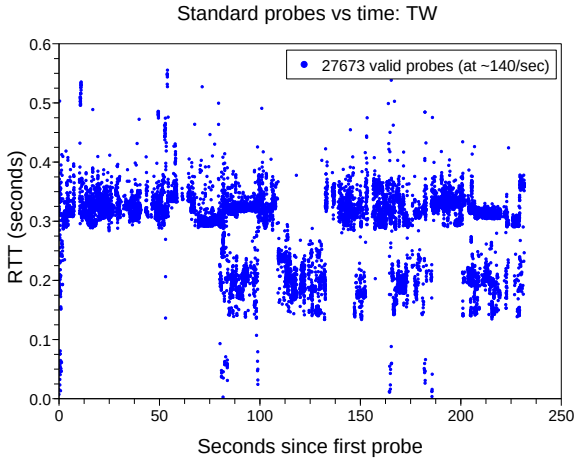

Figure 3: CS:S game server RTTs vs time as seen from Taiwan in Nov'07

- Present results from closer servers before those of more distant servers (as measured by RTT)
- Automatic early termination of search sequence (reducing the time spent during discovery)
- Function from behind consumer NAT devices, anywhere on the planet, without additional manual intervention or configuration by the player

The first goal reduces the time taken to locate game servers whose RTT is likely to be low enough for enjoyable and competitive online play. The second goal reduces network traffic generated during server discovery. The third goal is a usability issue. Although a growing number of NAT-enabled consumer routers allow private hosts to query the device's public IP address, this is by no means universal. It is beneficial to not rely on knowing one's public IP address.

3.1 Background

The challenge of finding FPS servers with low enough RTT is well recognised [6]. To date research has focused on relocating clients to optimally placed servers (e.g. [7]), 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. A master server cannot pre-sort the list of game servers in order of ascending RTT because it cannot know the network conditions existing between any given client and every game server. (A master server also cannot trust other clients to accurately report such information from different parts of the Internet - a few misbehaving players injecting corrupt RTT information could easily disrupt such a system.) Every client must emit active probes to establish their own sense of RTT to individual game servers.

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' [8]. First the client would map server IP addresses to their country of origin (for example, using MaxMind's free GeoLite Country database [9]). Then a selected subset of 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. Unfortunately, country codes are a very coarse indicator. The estimated RTT for particular countries often bore little relation to the spread of RTTs of individual servers falling under the same country code, limiting the scheme's ability to reliably determine when to automatically terminate a server discovery probe sequence.

3.2 Clustering by origin Autonomous System

There are three key steps to optimised server discovery: clustering, calibration, and optimised probing.

In [8] game servers were *clustered* according to their country of origin. This paper proposes clustering game servers by the Autonomous System (AS) to which each game server's IP address belongs (its *origin AS*). AS numbers are used in inter-domain routing (by the Border Gateway Protocol, BGP [10]) to identify topologically distinct regions of the In-

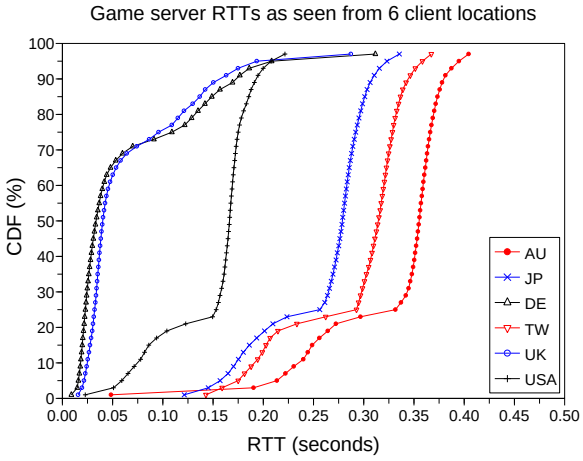

Figure 4: Distribution of measured CS:S game server RTTs seen from all six client locations

ternet. From the perspective of any given CS:S client, game server IP addresses sharing a common origin AS are likely to share a similar distance (and hence RTT) from the client.

Calibration involves probing a fraction of all registered game servers in each cluster to establish a reasonable RTT estimate with which to rank the entire cluster. It also ensures that rankings are based on the network conditions being experienced by each client and relative to each client's current location.

Optimised probing involves probing all the remaining game servers in order of ascending RTT of their cluster (excluding game servers already probed during the calibration phase - we already know their RTTs).

Algorithm 1 summarises the process, and details a further optimisation found to be beneficial. The topology within any AS may encompass servers exhibiting a modest range of RTTs. To detect such clusters we randomly select one of the initial N_{sample} servers from each unique /16 prefix present in the IP addresses making up the cluster. When the samples for a given $RTT_{cluster}$ are spread 'too wide' the cluster is split into multiple new (smaller) clusters, each one containing all the game servers falling under different /16 address prefixes within the original cluster. (Algorithm 1's threshold values - 20th and 80th percentiles differing by more than 40ms - were found to provide good calibration and ranking. Space limits preclude a more detailed discussion.)

3.3 Automatic termination of server discovery

Ranking by $RTT_{cluster}$ does not guarantee successive probes will see ascending RTTs during re-ordered probing. Nevertheless, the RTT averaged over recent probes will trend upwards and we may implement automatic early termination ('auto-stop') of re-ordered probing when the RTT trends above a player-specified threshold (e.g. 200ms). Auto-stop will reduce the number of probes emitted, the number of UDP flows generated, and the player's time waiting to know if all *playable* servers have been probed. (The latter is particularly relevant 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 describes a suitably cautious approach to determining when to auto-stop.

Algorithm 1 Calibration and probing of clusters

1. Retrieve all game servers and AS numbers
2. Every server's AS number is its *cluster_id*. Servers sharing a *cluster_id* are in the same cluster
3. **Calibration:** For each cluster_id,
   (a) $N_{sample} = \sqrt{N_{cluster}}$, where $N_{cluster}$ is the number of servers in the cluster
   (b) $S_{calibration} = N_{sample}$ servers randomly selected from the cluster, one from each /16 subnet present within the cluster
   (c) $RTT_{cluster}$ = median RTT of the cluster after probing each server in $S_{calibration}$
   (d) 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. Probe one previously un-probed server randomly selected from each new cluster. The probed RTT becomes $RTT_{cluster}$ for the new cluster
4. Rank every cluster in ascending order of $RTT_{cluster}$
5. **Re-ordered probing:** Probe all remaining game servers in order of their cluster's rank. Within a given cluster, probe servers in the order they were originally returned by the master server.

Today's modern FPS games require processor speeds in excess of 1GHz. The clock cycles required to implement Algorithms 1 and 2 are negligible relative to the time required to transmit the calibration and re-ordered probes. Because Algorithm 1 never sends more probes than conventional server discovery, the performance (time to discover acceptable servers) of Algorithms 1 and 2 working together will never be worse than the performance of conventional server discovery. (The worst case scenario is a client for whom all, or almost all, available game servers actually have an RTT less than RTT_{stop} - auto-stop might not be triggered before the client simply runs out of servers to probe.)

3.4 Mapping game servers to clusters

The information to map IP addresses to AS numbers is actively maintained by BGP-speaking devices (routers and non-routers) within every Internet service provider. A logical place to perform this mapping is the Steam master server. Open-source tools that can 'speak' BGP (such as Quagga [11]) are freely available and could be integrated into the Steam master server with little (conceptual) diffculty.

Currently the master server returns an unordered list S_1, S_2, ... S_N where S_x is a 6-byte <IP addr:port> pair. AS

Algorithm 2 Automatic termination of optimised probing

- RTT_{stop} - maximum RTT considered playable (e.g., $\text{RTT}_{stop} = 200\text{ms}$)
- $\text{W}_{autostop}$ - sampling window size (e.g., $\text{W}_{autostop} = 100$)
- Wait for $\text{W}_{autostop}$ servers to be probed
- Terminate probing when $\text{RTT}_{bottom} \geqslant \text{RTT}_{stop}$, where RTT_{bottom} is the RTT below which 2% of the last $\text{W}_{autostop}$ RTT samples have fallen

Client	AS Clusters	Calibration probes	Auto-stop	
AU	1240	3374	35.5 (sec)	14.8 (%)
DE	1182	3166	228.6 (sec)	97.3 (%)
JP	1176	3206	72.6 (sec)	31.3 (%)
TW	1173	3201	67.8 (sec)	29.0 (%)
UK	1176	3150	227.7 (sec)	97.8 (%)
USA	1173	3190	229.9 (sec)	99.1 (%)

Table 2: Optimised discovery from six locations

numbers may be embedded by instead returning a list of the form AS_1, S_{11}, S_{12}, ... S_{1N} , AS_2, S_{21}, S_{22}, ... S_{2N} ... and so on. Each AS_x indicates the origin AS to which the following game servers belong. The AS_x is encoded as 6-byte S_x field with the 'port' field set to zero (an otherwise invalid port for a genuine game server) and the non-zero AS number encoded in the 4-byte 'IP addr' field. (New AS numbers take 4 bytes. Traditional 2-byte AS numbers would be encoded as 4-byte numbers with the top 2 bytes zeroed.)

In principle clustering might also be performed at the client, but this would require all clients have access to up-to-date BGP routing information (impractical) and create additional network traffic for limited return.

4. ILLUSTRATING THE OPTIMISATION

Space constraints preclude an exhaustive analysis of section 3's client-side optimisation. Instead, the real-world datasets in Table 1 are used to illustrate the potential impact of a client applying Algorithms 1 and 2. Table 2 summarises some key details. 'AS clusters' is the number of distinct clusters created from the set of servers returned by the master server, 'Calibration probes' shows the number of probes sent during Algorithm 1's calibration phase, and 'Auto-stop' indicates how soon the optimised server discovery process terminated (in seconds and as % of worst-case).

Figures 5, 6 and 7 illustrate the impact on CS:S clients located in Australia, Japan and the UK respectively. For each client the optimised probe sequence generates traffic in two phases - 'calibration probes' and 're-ordered probes'. It takes ˜23 seconds in each case (˜3200 probes at 140/second) to calibrate each client and begin issuing re-ordered probes. Each figure shows the variation of RTT_{bottom} over time, up to the point where auto-stop occurs. The auto-stop RTT threshold is 200ms, and the estimated auto-stop time is indicated by a blue vertical line. Re-ordered probes are plotted beyond auto-stop to illustrate the effectiveness of re-ordering.

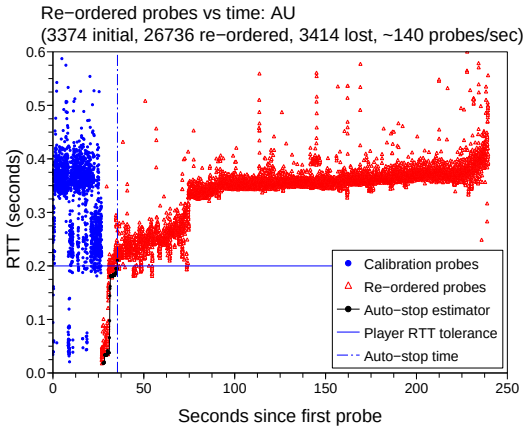

Figure 5: Optimised discovery, CS:S client in AU

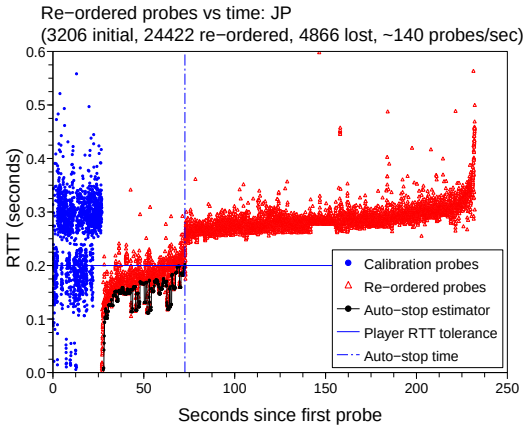

Figure 6: Optimised discovery, CS:S client in JP

The AU, JP and TW clients all see distinct improvements - auto-stop causes the probing to terminate well before the ˜230 second period taken by regular server discovery. (TW is similar to JP and thus not shown.)

The UK, USA and DE clients see a fairly neutral impact due to being close to large concentrations of game servers under 200ms (USA and DE are similar to the UK and thus not shown). Some servers over 200ms are seen towards the end of the probe sequence, but auto-stop errs on the side of continuing and the clients ultimately probes virtually all active servers.

With an optimised search order and auto-stop, clients far away from most game servers 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, the number of transient UDP flows in the network and the number of bytes sent or received.

5. ISSUES AND FUTURE WORK

Algorithm 2 relies on an inherently noisy signal (RTT_{bottom}), and may terminate prematurely if the cluster ranking is sufficiently mis-ordered, $\text{W}_{autostop}$ is too small or RTT_{stop} is set too low. Figure 5 shows that a small number of servers with an RTT under RTT_{stop} can be missed if auto-stop occurs early. Given the general uncertainty associated with con-

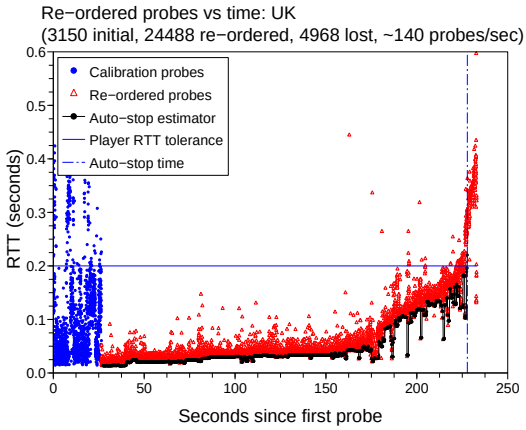

Figure 7: Optimised discovery, CS:S client in UK

ventional FPS server discovery this may be an acceptable weakness. RTT_{stop} might well be hardcoded into a client at 200ms without significant concern, yet if a client allows manual selection of the auto-stop threshold, RTT_{stop} should be set slightly higher than the player's RTT preference.

Calibration balances a need to probe as few servers as possible while constructing a useful $RTT_{cluster}$ for each cluster. Increasing N_{sample} increases the probability of detecting clusters that should be sub-divided (and thus constructing new clusters more accurately representing the RTTs of the cluster's members). The impact of alternative definitions for N_{sample} is a topic for future work.

Future work will also look at the impact of subdividing AS-based clusters on longer or shorter prefix lengths than /16 when the calibration phase detects too much spread in sampled RTTs, and the use of RTT spreads less than 40ms to trigger such subdivision.

As written, Algorithms 1 and 2 appear to have worked well for the datasets tested in this paper (and many similar datasets not reported here). Nevertheless, this paper should motivate a more detailed analysis of the trade-offs inherent in each algorithm.

Embedding AS numbers will increase the typical master server reply list by roughly 4% (˜1200 additional 6-byte AS_x markers, or 7Kbytes). However, there's a nett gain to the client if auto-stop eliminates ˜55 or more *A2S_INFO Reply* packets (which are often over 135 bytes long). For many clients auto-stop will eliminate hundreds or thousands of *A2S_INFO Request/Reply* probes.

Retrieving all game servers (and AS_x markers) from the master server before initiating calibration and re-ordered probing would be a change for Steam clients, but represents no change to server browsers such as qstat.

6. CONCLUSION

This paper proposes a novel method for improving the FPS game server discovery process, and illustrates the method's potential using examples based on Valve's Counterstrike:Source. Game servers from the same origin Autonomous System are clustered together for probing. Probes to a subset of the servers in each cluster are used to estimate the RTT to all servers in the cluster, and thus rank the clusters for final probing in ascending order of likely RTT. By automatically terminating re-ordered probing when RTTs start exceeding a player-nominated threshold, this method can reduce server discovery time and associated traffic down to less than 20% of conventional server discovery. The method adapts to wherever the client is located on the Internet, without player intervention. Clients who are distant from the majority of game servers receive the most benefit. Clients located close to most game servers experience performance (resource consumption and timeliness) no worse than that of conventional server discovery. A number of avenues for future work and improvement are identified.

This method may be applied to other online FPS games that perform server discovery like Counterstrike:Source, or utilised by non-game clients (such as automated services who wish to efficiently probe only game servers within a limited RTT radius). It can benefit any FPS game whose community is geographically diverse and where many clients and servers are separated by more than the RTT typically tolerated for competitive online FPS game-play.

7. ACKNOWLEDGEMENT

I am grateful to Mark Claypool for providing me with access to PlanetLab, and Geoff Huston for a BGP-feed to identify origin AS numbers.

8. REFERENCES

[1] G. Armitage, M. Claypool, and P. Branch, *Networking and Online Games - Understanding and Engineering Multiplayer Internet Games*. United Kingdom: John Wiley & Sons, Ltd., June 2006.

[2] Valve Corporation, *CounterStrike: Source*, http://counter-strike.net/, accessed February 8th 2008.

[3] ——, *Server Queries*, http://developer.valvesoftware.com/wiki/Server_Queries, as of February 7th 2008.

[4] *QStat*, http://www.qstat.org/, accessed February 8th 2008.

[5] PlanetLab, *PlanetLab - An open platform for developing, deploying, and accessing planetary-scale services*, https://www.planet-lab.org/, accessed February 8th 2008.

[6] M. Claypool, "Network characteristics for server selection in online games," in *ACM/SPIE Multimedia Computing and Networking (MMCN)*, January 2008.

[7] C. Chambers, W.-C. Feng, F. W.-C., and D. Saha, "A geographic, redirection service for on-line games," in *ACM Multimedia 2003 (short paper)*, November 2003.

[8] G. Armitage, C. Javier, and S. Zander, "Topological optimisation for online first person shooter game server discovery," in *Proceedings of Australian Telecommunications and Network Application Conference (ATNAC)*, December 2006.

[9] MaxMind, *GeoLite Country*, http://www.maxmind.com/app/geoip_country, accessed February 8th 2008.

[10] Y. Rekhter, T. Li, and S. Hares, "RFC 4271: A Border Gateway Protocol 4 (BGP-4)," Jan. 2006. [Online]. Available: http://tools.ietf.org/html/rfc4271

[11] *Quagga Software Routing Suite*, http://www.quagga.net/, accessed February 8th 2008.