Further Optimising Online FPS Game Server Discovery

Christopher Leong[1]
Centre for Advanced Internet Architectures. Technical Report 090911A
Swinburne University of Technology
Melbourne, Australia
ckcleong@gmail.com

***Abstract*- This paper proposes several improvements to a proposed game server discovery algorithm based on clustering servers by origin Autonomous System (AS). Following a brief overview of current game server discovery processes, the algorithms on which this paper is based are outlined. Using Valve's Counterstrike:Source game and datasets employed in the previous algorithm's paper, several improvements to the sub-clustering and calibration algorithms are explored. This includes sub-clustering based on a dynamic choice of network prefix and alternative definitions for a function to choose the number of calibration probes. The suggested enhancements allow a significant reduction in the number of calibration probes required while maintaining reasonable reordering of servers. Subsequently, appreciable reductions in the time and network traffic required over the existing proposed algorithms are observed.**

Keywords- Server discovery, search optimisation, latency estimation

I. Introduction

Many first person shooter (FPS) games facilitate multiplayer competition over data networks. These games typically employ a client-server topology and each server hosts anywhere from 4 to around 30+ players at any one time[1]. As these servers may be operated by anyone from large internet service providers (ISPs) to individual enthusiasts, there are many servers players may potentially join. The aim of game clients is to present an up-to-date list of active game servers so that a player can select a suitable server on which to play.

The act of creating this list of up-to-date active game servers is known as server discovery. Game clients typically query a central database (known as a master server) where a list of currently available game servers is maintained. Using this list, they then proceed to sequentially query each game server to generate a local list of information regarding each server. This may include the number of players currently playing on the server, the game/map type and the round trip time (RTT, commonly referred to as 'lag'). Based on this locally generated list, a player proceeds to select a suitable server on which to play.

With the fast paced and reaction based action that typically characterises FPS style games, latency plays an important role in ensuring fair and competitive gameplay. It has been shown that competitive online FPS requires latencies below 200ms[1]. Here the main issue with server discovery arises: players wishing to join low latency servers must wait for the server discovery process to complete to ensure that all potentially suitable game servers are listed in their game client. Coupled with the large number of servers available, this process proves time consuming, resource wasting and inefficient; players query thousands of servers while generating megabytes of network traffic to effectively join a single server.

This paper explores potential improvements to a previously proposed optimised server discovery algorithm based on clustering servers by origin Autonomous System (AS)[2]. While highly effective in reducing the time and amount of traffic generated in the server discovery process, potential areas of improvement were suggested by the author. These include a reduction in the number of required calibration queries/probes and alternative sub-clustering algorithms. Both of these areas have the potential to further improve the accuracy in reordering servers with fewer required initial samples and hence reduce the overall time required for the server discovery process. The same dataset employed in [2] and Counterstrike:Source are again used as an illustrative example of an online FPS game.

The rest of the paper is organised as follows. A brief overview of the server discovery process using Counterstrike:Source (CS:S), current solutions to improving the server discovery process and the proposed optimised server discovery process based on clustering by AS are outlined in section 2. This is followed by an exploration of the issues raised by the author of this algorithm in section 3. Section 4 provides a demonstrative illustration of the combined effects of the optimisations provided in this paper. Areas identified with potential for further research and improvement are outlined in section 5. The paper concludes in section 6.

II. Server Discovery

This section provides a brief overview of current game server discovery methods using Counterstrike:Source as an illustrative example. It then outlines some of the currently implemented solutions aimed at reducing the resources used during server discovery and their downfalls. Finally, a summary of the proposed optimised server discovery process based on clustering by AS is given.

1 This author is currently an engineering student at The University of Melbourne. This report was written during the author's winter internship at CAIA in 2009.

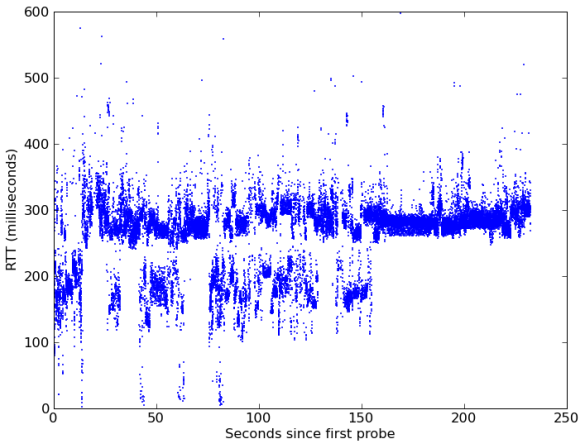

Figure 1 CS:S game server RTTs vs time as seen from Japan

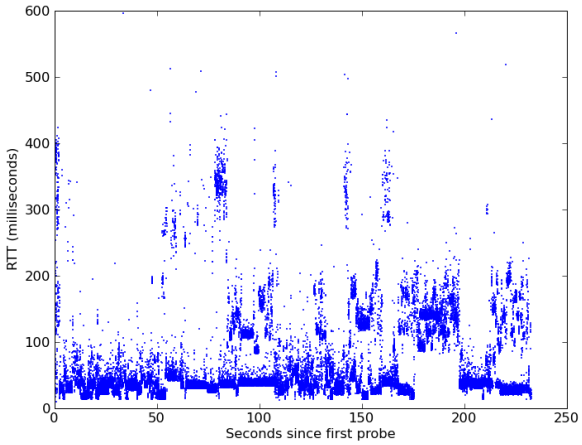

Figure 2 CS:S game server RTTs vs time as seen from the UK

A. *Counterstrike:Source*

Game server discovery typically occurs in two phases:

1. A game client queries a master server for a list of the addresses of currently available game servers
2. The game client then queries each of the addresses identified in the previous step sequentially to determine information such as the number of players currently participating in the game and the game/map type. An estimate of the latency between the client and server is also usually inferred through this query and response.

The protocols used when querying the master server and the subsequent game servers for CS:S are well documented on the Valve developer website[3]. In the case of CS:S, the two phases involve:

1. Multiple queries to the master server to generate a list of addresses of currently available game servers. Each request contains information regarding the broad geographical region in which you wish to find servers, an identifier for the next packet required from the master server and a filter which tells the master server to return only servers running a particular game, map, etc.
2. Game clients then proceed to query each of the servers previously identified sequentially using the A2S_INFO[4] query type. A2S_INFO query responses from the game servers detail information such as the number of players currently participating in the game and the game/map type.

B. *Issues with current server discovery methods*

To illustrate the issues regarding current game server discovery mechanisms, real-world server discovery data is used. The dataset used throughout this paper was gathered using qstat[5] to probe all available CS:S servers using clients at various Planetlab nodes[6]; it is identical to the dataset employed in [2]. To remain consistent with [2], a nominal rate of 140 probes per second is again employed.

Figure 1 and Figure 2 illustrate the second, most time consuming phase of the server discovery process for two particular game clients in different regions. These are Japan and the United Kingdom respectively. The Japanese client was chosen as an example of a client that is distant to many low RTT servers while the client from the UK was chosen as an example of a client that is close to many low RTT servers.

It is clear from the probe distributions that, regardless of the game client's distance to many low latency servers, the discovered game server RTTs fluctuate across the entire server discovery period. Thus a player must wait for all game servers to be probed before they can conclude that all 'playable' servers have been located.

C. *Existing solutions*

There are currently two methods employed by game clients to assist players with finding suitable game servers. These include server-side and client-side filtering.

Basic server-side filtering is provided when querying the master server. The query packet allows the specification of the broad geographical region and desired game/map type of servers. This information is used by the master server to reduce the number of returned game server addresses. Hence, the number of subsequent game servers to be queried is reduced. This lessens the time spent and unnecessary network traffic generated searching for a suitable game server.

Client-side filtering allows game clients to specify certain attributes which may be used to sort or remove servers from the game browser's current view. However, while this may aid a player in searching for a suitable game server on which to play, it does not affect the total time and network traffic required to gather information about relevant servers; it merely improves the game browser's presentation of current information to the player.

D. *Proposed solution*

With the aim of presenting lower RTT servers before higher RTT servers, automatic early search termination given a desired threshold and being able to function with minimal player configuration, a method involving clustering by AS was devised[2]. To overcome the chicken-and-egg problem of wishing to probe by increasing RTT but needing to probe a game server for its RTT to be able to sort it by RTT, the algorithm works in three key steps:

1. Clustering. The game servers returned by the master server are grouped by its origin AS. The rationale behind this is that different autonomous systems identify topologically distinct regions of the Internet. Hence, they should share a similar RTT from the client.

Algorithm 1: Calibration and probing of clusters

1. Retrieve all game servers and AS numbers
2. Cluster servers by AS number
3. For each AS cluster,
   a) $N_{sample} = N_{cluster}^{1/2}$ where $N_{cluster}$ is the number of servers in the same cluster
   b) Probe N_{sample} randomly selected servers from the cluster, one from each /16 subnet present within the cluster
   c) $RTT_{cluster}$ = median RTT of the N_{sample} sampled servers in the cluster
   d) If the 20th and 80th percentile measurements of all the sampled servers differ by more than 40ms:
      i. Split the members of the cluster into new clusters
      ii. Each new cluster consists of members of the old cluster who belong to the same /16 subnet
      iii. Probe one previously un-probed server from each new cluster. The probed RTT becomes $RTT_{cluster}$ for the new cluster
4. Rank all clusters in order of ascending $RTT_{cluster}$
5. Probe all remaining 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

2. Calibration. A subset from each AS is probed and the results used to infer a reasonable estimate of the RTT for the AS as a whole. The calibration probes are also used to determine whether hosts within an AS display clustering and should be further broken up for re-ordering.
3. Optimised probing. This involves querying the remaining servers in order of sorted cluster using the information gathered during calibration. Servers within clusters are probed in the order originally returned by the master server.

The above process is summarised in Algorithm 1.

To implement automatic early search termination in the server discovery process, another algorithm was proposed by the same author[2]. Since the previous algorithm does not guarantee that successive probes will have ascending RTT (however, successive probes should trend upwards), a method that employs a sorted window of current probes is used to determine a suitable stop time. This method is described in Algorithm 2.

Figure 3 and Figure 4 illustrate the effects that these algorithms have on the server discovery process. The two distinct phases of calibration and reordered probing are shown in blue and red respectively. A 200ms threshold for 'playable' servers is shown as a horizontal dashed line. The autostop algorithm's RTT_{bottom} is shown as the black line, while the autostop termination time is shown as the dashed vertical line.

Algorithm 2: Automatic termination of optimised probing

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

To illustrate the effectiveness of re-ordering, probes beyond the autostop termination time are also plotted. A sorted list of 'probes to be re-ordered' is also superimposed on the re-ordered probes in green. This represents the ideal case in which all remaining game servers' RTTs are known a priori and provides a visual benchmark. These colour and plot conventions will be used throughout the rest of this paper.

It can be seen that the most tangible improvements are achieved by clients which are distant to many low RTT servers. The reordering algorithm effectively allows the game client to query lower latency servers first while the autostop algorithm appropriately terminates the discovery process when no more 'playable' servers are expected to be found. In the case of this particular Japanese game client, only 32% of the total server discovery time was needed.

Clients which are closer to many low RTT servers see a fairly neutral impact in server discovery time and traffic generated. Since the majority of servers are already below the 'playable' threshold, the autostop algorithm stops the discovery process late in the process. However, since the game servers are still reasonably re-ordered, lower autostop thresholds may be employed to greater effect.

III. Issues and potential improvements

As seen in the previous section, the currently proposed optimised server discovery algorithm is highly effective at reducing the time and amount of traffic generated during the server discovery process. However, the author suggested several areas that could possibly be improved. These include alternative methods for determining the number of required calibration queries/probes and alternative sub-clustering algorithms. Both of these areas have the potential to further improve the accuracy in reordering servers with fewer required initial samples and hence reduce the overall time required for the server discovery process. These issues will be explored in the following subsections.

A. An alternative sub-clustering algorithm

The current algorithm for sub-clustering involves clustering autonomous systems into /16 networks if the 20th and 80th percentile measurements for the cluster differ by more than 40ms. The impact of dividing the network using different prefix lengths is explored through the following proposed change to the prefix choice in the current algorithm.

The proposed change involves identifying the number of subnets an autonomous system can be broken up into using different prefix lengths. As the prefix length is increased, the autonomous system is broken up into more, smaller subnets.

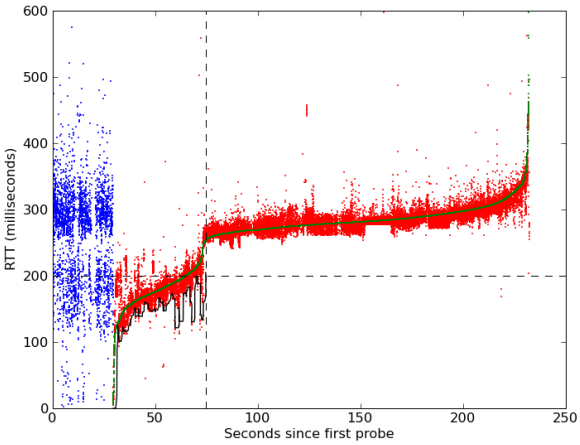

Figure 3 Optimised discovery, CS:S client in Japan

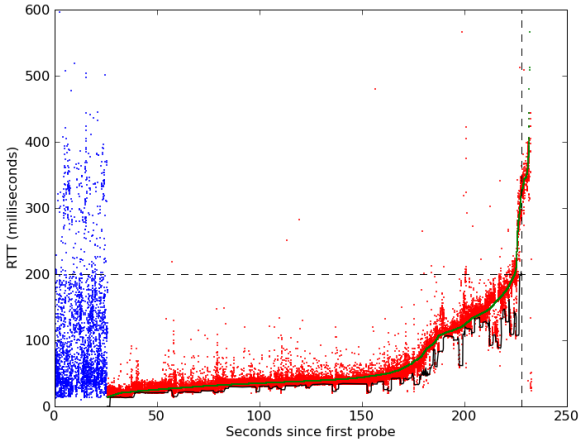

Figure 4 Optimised discovery, CS:S client in UK

N_{sample}	Calibration probes	Autostop (time and % worst case)		% all probes $<RTT_{stop}$ found
$(N_{cluster})^{0.5}$	3185	74.5s	32.1%	100.0%
$(N_{cluster}/2)^{0.5}$	2391	70.9s	30.5%	99.9%
$(N_{cluster}/4)^{0.5}$	1908	68.3s	29.4%	99.7%
$(N_{cluster}/8)^{0.5}$	1623	66.1s	28.5%	99.6%

Table 1 Scaling $N_{cluster}$, CS:S client in Japan

N_{sample}	Calibration probes	Autostop (time and % worst case)		% all probes $<RTT_{stop}$ found
$(N_{cluster})^{0.5}$	1911	68.6s	29.5%	99.7%
$(N_{cluster}/2)^{0.5}$	1680	67.1s	28.9%	99.7%
$(N_{cluster}/4)^{0.5}$	1524	66.1s	28.5%	99.7%
$(N_{cluster}/8)^{0.5}$	1413	65.3s	28.1%	99.6%

Table 2 Prioritised sampling, CS:S client in Japan
(N_{sample} = 1 for $N_{cluster}$ < 100)

Then, based on the choice of N_{sample}, the choice of prefix length is chosen such that the autonomous system is broken up into N_{sample} or less subnets.

The rationale behind this method is that given a desired number of samples, sampling based on a fixed prefix length may lead to some subnets being probed multiple times while other subnets may not get probed at all. Choosing a prefix based on the desired number of samples ensures that given the number of desired samples, probes from as many possible distinct clusters in the address space are sampled. This process is described in Algorithm 3.

Figure 5 illustrates the effect that this algorithm has on the server discovery process for the client in Japan. Contrasting to Figure 3, it can be seen that a marginal reduction in the time required is achieved with negligible loss in the number of 'playable' servers found before termination. More importantly, the reordering of servers is more consistent. This is especially evident towards the beginning of the optimised probing step; groups of servers which should have been probed later in the process are relocated correctly.

Figure 6 and Figure 4 contrast the effect that this algorithm has on the client in the UK. The effect on re-ordering is more subtle than in the case of Japan. It can be seen that small clusters of servers around 200 seconds into the optimised discovery process are re-ordered appropriately with this alternative algorithm.

Although the direct benefits of this alternative sub-clustering algorithm may appear marginal, in the next section it is shown that this technique, combined with an alternative choice in the number of calibration probes, yields appreciable gains.

B. Alternative choices in the number of calibration probes

The number of samples to be taken from each AS cluster for calibration is currently based on the square root of the number of game servers found within an AS. This seemingly arbitrary choice of the number of samples is investigated through the exploration of different functions used to define N_{sample}. Also, an alternative method in choosing N_{sample} is proposed and investigated.

To determine the sensitivity of the number of probes to the accurate reordering of clusters, different choices of N_{sample} were investigated. This was first achieved through scaling the number of autonomous systems used in the calculation of N_{sample} while retaining the original square root function.

A summary of the results from the client in Japan can be found in Table 1. It can be seen that even with as few as 50% of the original number of sampled probes, the optimised server discovery algorithm is able to remain highly accurate. In all instances over 99% of the total number of 'playable' servers were discovered before termination. Similar results were observed for the client in the UK: reducing $N_{cluster}$ by a factor of 8 saw a 50% fall in required calibration probes while still discovering nearly 98% of the total number of 'playable' servers.

An alternative method for choosing N_{sample} was also investigated. It is based on prioritising the sampling of larger autonomous systems. The rationale behind this decision is that servers within smaller AS clusters are more likely to be closer together (and hence share similar RTTs from the player). Hence, only a single probe is used to characterise smaller AS clusters while existing methods for the choice of N_{sample} are retained for processing larger clusters. The choice of 100 game servers within an AS was found to provide a good threshold (Space limits preclude a more detailed discussion of this choice).

Again, a similar trend emerges for the Japanese client as seen in Table 2. It can be seen that even without sub-clustering smaller autonomous systems, savings in the number of

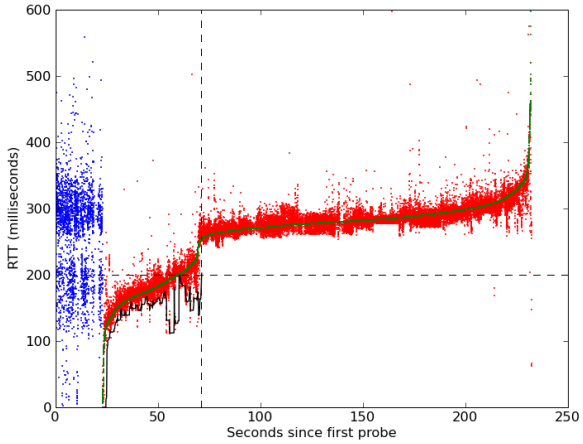

Figure 5 Alternative sub-clustering algorithm, CS:S client in Japan

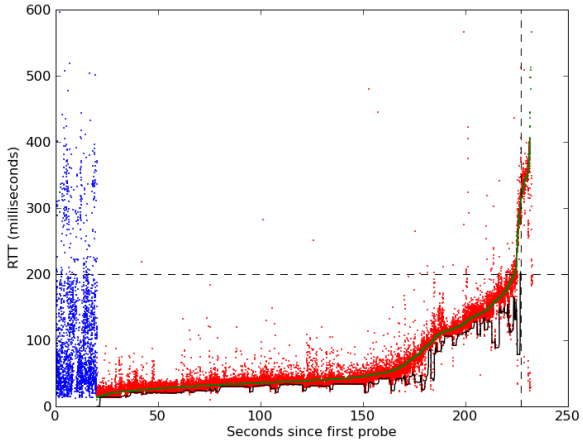

Figure 6 Alternative sub-clustering algorithm, CS:S client in UK

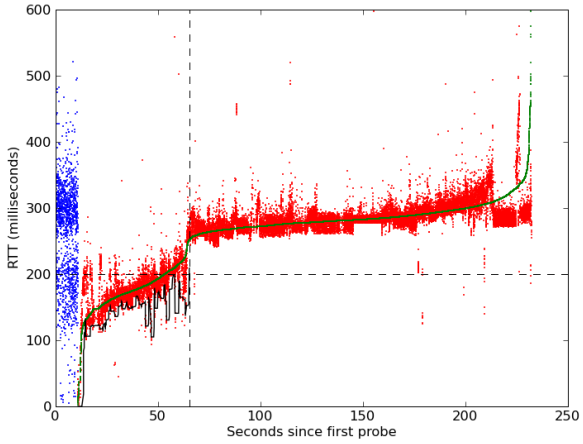

Figure 7 Reordering issues arise with using fewer calibration probes, CS:S client in Japan

calibration probes required and the overall time taken are achieved with a minimal effect on the number of detected 'playable' servers. Similar results were observed with the UK client.

A consequence of using fewer calibration probes that is not immediately evident in the above tables is the effect on reordering. Even though the percentage of servers found with desirable latencies remained highly consistent, as the number of calibration probes was reduced, Figure 7 illustrates the undesired effect of poorer reordering. It can be seen that, especially towards the end of the complete discovery process, that some clusters of servers are sub-optimally re-ordered. In the next section, it will be shown that a combination of the above-mentioned techniques improves the situation.

Algorithm 3: Alternative algorithm for calibration and probing of clusters

1. Retrieve all game servers and AS numbers
2. Cluster servers by AS number
3. For each AS cluster,
   a) $N_{sample} = N_{cluster}^{1/2}$ where $N_{cluster}$ is the number of servers in the same cluster
   b) N_{actual} = the number of subnets $\leq N_{sample}$ resulting from breaking up the cluster using an increasing prefix length
   c) P_{actual} = prefix found in previous step
   d) Probe N_{actual} randomly selected servers from the cluster, one from each subnet whose prefix is P_{actual}
   e) $RTT_{cluster}$ = median RTT of the N_{sample} sampled servers in the cluster
   f) If the 20th and 80th percentile measurements of all the sampled servers differ by more than 40ms:
      i. Split the members of the cluster into new clusters
      ii. Each new cluster consists of members of the old cluster who share the same subnet prefix P_{actual}
4. Rank all clusters in order of ascending $RTT_{cluster}$
5. Probe all remaining 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

IV. Illustrating the Combined Optimisations

To illustrate the combined optimisations suggested in this paper, data from both the Japanese client and the UK client in Table 3 and Table 4 respectively are used to highlight several results of interest.

'Unmodified' represents the results from using the original optimised server discovery algorithm (described in Algorithm 1 and 2). The results from using a combination of the alternative sub-clustering algorithm (described in Algorithm 3) and a reduced number of calibration probes are referred to as 'modified'. The reduced number of calibration probes is based on prioritising the sampling of larger autonomous systems ($N_{sample} = (N_{cluster}/8)^{0.5}$ for clusters with >100 servers, $N_{sample} = 1$ otherwise). Finally, results from the extreme case of only randomly probing each AS cluster once are reproduced for comparison.

For the client in Japan, it can be seen that using only 43% of the originally required calibration probes, the combined modifications to the existing algorithm allowed for a reduction

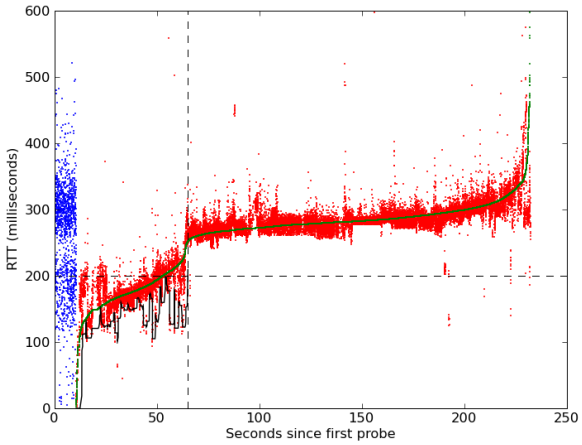

Figure 8 Plot of 'modified' algorithm, CS:S client in Japan

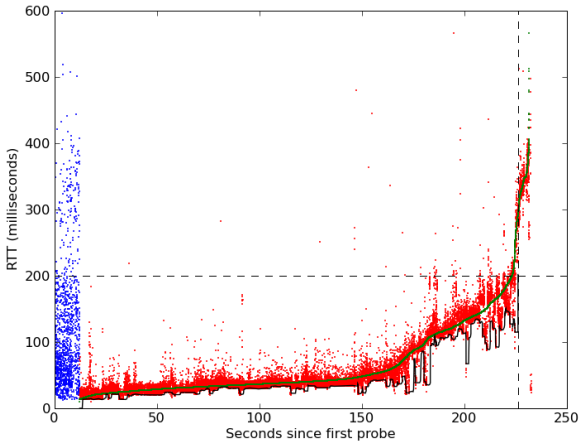

Figure 9 Plot of 'modified' algorithm, CS:S client in UK

Algorithm	Calibration probes	Autostop (time and % worst case)		% all probes <RTT_{stop} found
Unmodified	3185	74.5s	32.1%	100%
Modified	1362	65.1s	28.1%	99.6%
1 probe per AS	1176	55.8s	24.0%	84.8%

Table 3 Illustration of combined optimisations, CS:S client in Japan

Algorithm	Calibration probes	Autostop (time and % worst case)		% all probes <RTT_{stop} found
Unmodified	3136	227.7s	97.9%	100%
Modified	1366	226.2s	97.3%	100%
1 probe per AS	1176	226.2s	97.3%	100%

Table 4 Illustration of combined optimisations, CS:S client in UK

of 13% in the required time to complete the discovery process. This was achieved with a negligible loss in 'playable' game servers found before termination. Comparing to the extreme case, it can be seen that considerable results with only 15% more probes than absolutely required can be achieved.

While a similar reduction in the number of calibration probes was seen for the client in the UK, a time reduction of only 0.7% was observed. This was due to the majority of servers being under the 'playable' RTT threshold. However, it can be seen that the modified algorithm still managed to find all 'playable' servers using fewer calibration probes.

Figure 3 and Figure 8 contrast the original and modified algorithm's performance for the client from Japan while Figure 4 and Figure 9 contrast the original and modified algorithm's performance for the client from the UK. In both cases it can be seen that despite the large reduction in the number of calibration probes used in the modified algorithm, reasonable accuracy in reordering is still achieved.

V. Issues and future work

While the datasets employed in this paper demonstrated promising results, greater real-world testing on different connections from different regions should be investigated. This is to ensure that the underlying assumptions made were not unique to the particular game client data used.

A theoretical lower limit of probing only one server per AS to reorder clusters was mentioned when illustrating the improvements achieved through modifying the originally proposed algorithm. Exploring the possibility of using the AS number of the hop prior to a server's AS to cluster autonomous systems may prove a possible avenue to lower this limit.

Several other possible areas to explore include using different conditions and thresholds for sub-dividing clusters.

VI. Conclusion

With the large number of game servers available for players to join, the simple 'brute force' methods for server discovery are proving increasingly detrimental to the player experience. Having outlined several issues pertaining to current methods, a previously proposed process involving clustering by origin Autonomous System was investigated. While effective at reducing both the time and network resources required for the discovery process, several potential areas of improvement were explored.

This paper has shown that using alternative choices in sub-clustering and the number of calibration probes employed in the previously proposed optimised game server discovery process can appreciably improve the efficiency of the algorithm. Sub-clustering based on a dynamic choice of network prefix improved the re-ordering of clusters, displaying greater intelligence in the choice of calibration probes. Using a reduced number of calibration probes and concentrating calibration resources on larger, more potentially diverse AS clusters proved viable but demonstrated a reduction in reordering accuracy. However, the modifications proved to complement each other's weaknesses to synergistically yield appreciable reductions in the amount of time and network traffic generated.

Acknowledgements

I would like to thank Jason But and Philip Branch for their guidance and assistance over the course of this project, Grenville Armitage for providing the opportunity to undertake this internship and CAIA for their warm hospitality throughout my visit.

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] G. Armitage, Optimising Online FPS Game Server Discovery through Clustering Servers by Origin Autonomous System. ACM NOSSDAV 2008, May 2008.

[3] Master Server Query Protocol, http://developer.valvesoftware.com/wiki/Master_Server_Query_Protocol, as of July 2009

[4] Server Queries, http://developer.valvesoftware.com/wiki/Server_Queries, as of July 2009

[5] QStat, http://www.qstat.org/, accessed July 2009

[6] PlanetLab, PlanetLab - An open platform for developing, deploying, and accessing planetary-scale services, https://www.planet-lab.org/, accessed July 2009