This full text paper was peer reviewed at the direction of IEEE Communications Society subject matter experts for publication in the WCNC 2010 proceedings.

Minimising Disruption Caused by Online FPS Game Server Discovery in a Wireless Network

Jason But, Christopher Leong, Philip Branch and Grenville Armitage
Centre for Advanced Internet Architectures
Swinburne University of Technology
Melbourne, Australia
Email: jbut@swin.edu.au, ckcleong@gmail.com, {pbranch,garmitage}@swin.edu.au

***Abstract*—A key part of First Person Shooter network gaming is the game server discovery phase. Whilst probing for suitable servers from a wireless network, a large burst of network traffic is generated, potentially leading to detrimental effects on network capacity available to other wireless users. This can be minimised using an optimised algorithm to order the discovery probes and subsequently terminate the discovery process early. In this paper we explore further modifications to a previously proposed algorithm and examine its efficacy in further reducing the probe time/traffic during the server discovery phase. We show that it is possible to further reduce the overall discovery process duration by up to 13% while still presenting all suitable servers to the user for selection.**

I. Introduction

As access to wireless networks becomes more prevalent in the community, more people are using public wireless access points to conduct online activities. Amongst the online activities subscribed to users is online gaming. As the performance of wireless networks improves, increasing numbers of users will migrate to playing online games using wireless technology [1], [2].

First Person Shooter (FPS) games are amongst those with the most dependence on network performance in terms of delay and jitter [3]. There exists prior work in evaluating the performance of FPS games over wireless networks, but these examine the performance during gameplay [1], [2]. What is often not considered is how network game traffic behaves during the server discovery phase [4].

FPS games typically employ a client-server topology where each server hosts anywhere from 4 to around 30+ players at any one time [3]. 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. Server discovery covers the task of presenting an up-to-date list of active game servers so that a player can select a suitable server on which to play.

The process involves querying a master server which maintains a list of currently available game servers. These servers are then sequentially probed to determine game information and network latency [5]. A server is selected based on this information.

Latency plays an important role in ensuring competitive gameplay – competitive online FPS requires latencies below 200ms [3]. The issue with server discovery is that players wishing to join low latency servers must wait for the discovery process to complete to ensure that all potentially suitable game servers are listed for selection [6], [7].

The current sequential probing process is both time consuming and generates large bursts of network traffic. These bursts of network traffic can have adverse effects on the estimation of latency to individual game servers and impact on the network resources available to other users of the wireless medium [8]. To minimise this impact, we should aim to reduce the number of game servers probed.

This paper explores potential improvements to a previously proposed optimised server discovery algorithm based on clustering servers by origin Autonomous System (AS) [9]. These include a reduction in the number of calibration probes and alternative sub-clustering algorithms. We will examine the efficacy of both approaches in further reducing overall time – and network traffic – required for the server discovery process. We use the same dataset employed in [9] – consisting of Counterstrike:Source traffic – as an illustrative example of an online FPS game.

The rest of the paper is organised as follows. In Section 2 we present an overview of some of the issues of FPS games in a wireless environment. This is followed by a discussion on existing work on how game server discovery works, some of the issues involved, and currently proposed solutions in Section 3. Section 4 examines techniques to further reduce the probe time whilst increasing the accuracy of probe reordering. The paper concludes in section 5.

II. FPS Games in a Wireless Environment

Much prior work on running FPS games in a wireless environment has centred on how well wireless networks cope with network traffic generated by the online games, how that traffic inteferes with other wireless network users, and how scheduling schemes could improve the performance for game traffic [1], [2], [10]. Their results show that as long as the number of concurrent players is not too high, wireless networks can successfully run FPS games. However this work is constrained to traffic generated while *playing* the FPS game. During this phase, the traffic typically consists of small, regular UDP packets between each client and game server [3].

Little work has been done on the impact of game server discovery on a wireless network. During this phase, each

978-1-4244-6398-5/10/$26.00 ©2010 IEEE

game client generates single packet request-response flows to potentially thousands of different game servers over a short period of time [4], [5]. This burst of traffic to possibly large numbers of different IP addresses has the potential to impact heavily on network performance experienced by other users. Game server discovery traffic has the potential to tie up network resources for extended periods of time. Typical discovery probes might typically consist of ∼50 byte packets transmitted at a rate of ∼140 packets/s [11]. For server counts of ∼27,000 as witnessed in our captured traces, this procedure could take up to four minutes to complete.

Existing studies of non-reactive traffic (UDP/ICMP) over wireless networks demonstrate that the presence of regular frequency, non-reactive traffic can *steal* significant bandwidth from other – TCP-based – wireless network users [8].

Armitage [4] also highlights a second issue where the rate of server probing can impact on the accuracy of latency estimation by the client. This effect can be exarcerbated in a wireless network where increased contention decreases the total available slots for data transfer.

A final potential issue is one of access through a NAT. In the case of the wireless infrastructure provided by a small enterprise to the public, we expect cost to be one of the foremost considerations. As such, it is likely that wireless clients will be assigned private IP addresses and be situated behind a NAT. This could be implemented using low-cost, consumer grade equipment which may have limited resources to track the state of each connection through the NAT.

Game server discovery flows are short lived, yet the NAT will typically maintain state information for each of these flows for a number of minutes after the flow terminates. If the client is generating thousands of probes over an interval of a few minutes, this could limit the ability of the NAT to support other network flows.

A scheme that can sort game server probes based on their likely latency to the game client, and then terminate the probe phase early has the potential to minimise the impact of the game client on the wireless network. This can significantly decrease the generated network traffic and subsequently decrease the contention for the limited resources in the wireless environment.

III. Prior Work

This section provides a brief overview of current game server discovery methods using Counterstrike:Source (CS:S) 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 [9] based on clustering by AS is given.

A. Counterstrike:Source

The protocols used when querying the master server and the subsequent game servers for CS:S are well documented [12]. Game server discovery (as illustrated in Figure 1) typically occurs in two phases:

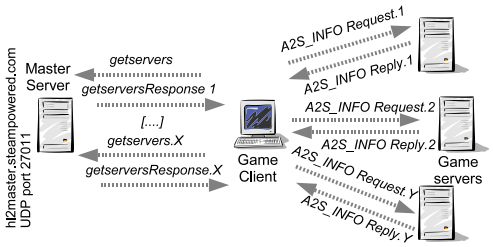

Fig. 1. Game Server Discovery Process

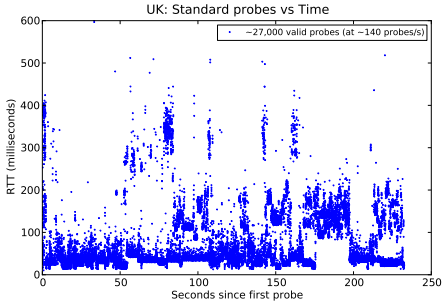

Fig. 2. CS:S game server RTTs vs time as seen from the UK

1) A game client queries a master server for a list of the addresses of currently available game servers
2) The game client then sequentially probes each game server to determine game information and an estimate of the latency between the client and server

B. Issues with current server discovery methods

To illustrate the issues regarding current game server discovery mechanisms, real-world server discovery data was used. The dataset used throughout this paper is identical to that employed in [9] and was gathered using **qstat**[1] to probe all available CS:S servers using clients at various PlanetLab nodes[2]. As in [9], we assume a nominal rate of 140 probes per second.

For our work, we examine server discovery for two game clients in different regions. A Japanese client was chosen as an example of a client that is distant to many servers. Conversely, a UK-based client is close to many servers.

Figure 2 illustrates the second, most time consuming phase of the server discovery process for the client in the UK. Servers are probed in the same order as presented by the master server. It is clear from the probe distributions that this list is not optimally ordered – a player must wait for all game servers to be probed before they can conclude that all *playable* servers have been located. A similar structure is seen for the Japanese client.

[1]http://www.qstat.org
[2]http://www.planet-lab.org

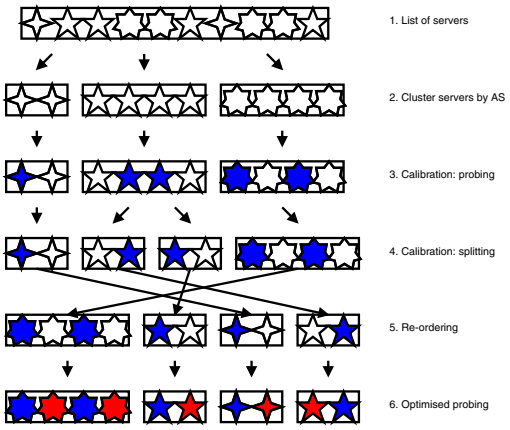

Fig. 3. Server discovery cluster/calibrate/probe algorithm

C. Filtering Servers

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.

Server-side filtering is provided when querying the master server and 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. As a smaller number of game servers are now queried, this decreases any 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, it does not affect the total time and network traffic required to gather information about relevant servers.

D. Reducing Probe Traffic

Armitage [9] proposed an optimised probing algorithm in which game servers are clustered by AS and then probed in increasing RTT order. This algorithm requires minimal user configuration and uses a desired RTT threshold to terminate game server probing. The probe reordering algorithm, illustrated in Figure 3, works in three key steps:

1) **Clustering:** The game servers returned by the master server are grouped by their origin AS. The rationale is that different autonomous systems identify topologically distinct regions of the Internet and should share a similar RTT from the client
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 sub-divided
3) **Optimised probing:** The remaining servers are queried 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

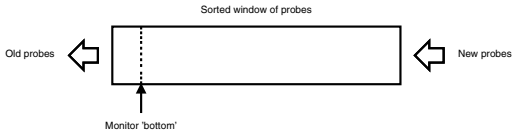

Fig. 4. Early probe termination algorithm

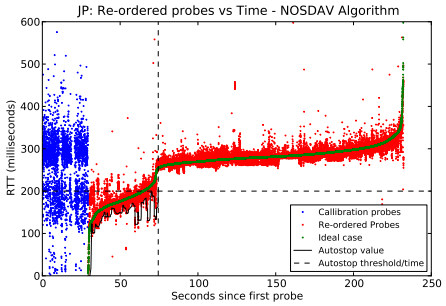

Fig. 5. Optimised discovery, CS:S client in Japan

Armitage also proposed a second algorithm to allow automatic early search termination [9]. The clustering algorithm does not guarantee that successive probes will have ascending RTT (although successive probes do trend upwards). The method, illustrated in Figure 4, employs a sorted window of recent probes, and stops when the monitored sample is higher than a nominated *playable* threshold.

Figures 5 and 6 recreate the work from [9] and 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 current RTT_{bottom} value of the autostop algorithm is shown as the black line, while the autostop termination time is shown as a dashed vertical line.

To illustrate the effectiveness of reordering, probes beyond the autostop termination time are also plotted. A sorted list of "probes to be reordered" is superimposed in green, representing the ideal case in which all remaining game servers' RTTs are known a-priori, thus providing a visual benchmark. These colour and plot conventions are used throughout the paper.

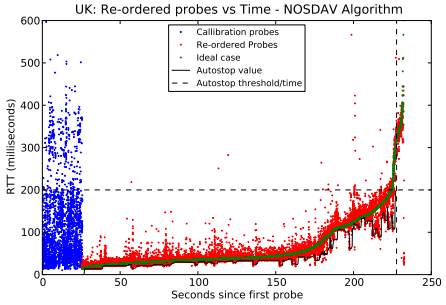

Fig. 6. Optimised discovery, CS:S client in the UK

TABLE I
SCALING $N_{cluster}$, CS:S CLIENT IN JAPAN

Algorithm	Calibration Probes	Autostop (time and % all probes)		% all probes $< RTT_{stop}$ found
$\sqrt{N_{cluster}}$	3,185	74.5s	32.1%	100%
$\sqrt{N_{cluster}/2}$	2,391	70.9s	30.5%	99.9%
$\sqrt{N_{cluster}/4}$	1,908	68.3s	29.4%	99.7%
$\sqrt{N_{cluster}/8}$	1,623	66.1s	28.5%	99.6%

TABLE II
PRIORITISED SAMPLING, CS:S CLIENT IN JAPAN SCALING ($N_{sample} = 1 for N_{cluster} < 100$),

Algorithm	Calibration Probes	Autostop (time and % all probes)		% all probes $< RTT_{stop}$ found
$\sqrt{N_{cluster}}$	1,911	68.6s	29.5%	99.7%
$\sqrt{N_{cluster}/2}$	1,680	67.1s	28.9%	99.7%
$\sqrt{N_{cluster}/4}$	1,524	66.1s	28.5%	99.7%
$\sqrt{N_{cluster}/8}$	1,413	65.3s	28.1%	99.6%

The most tangible improvements are achieved by clients which are distant to many 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 servers see a neutral impact in server discovery time and generated traffic. As the majority of servers are already below the *playable* threshold, the discovery process is stopped later. However, since the game servers are still reasonably reordered, lower autostop thresholds may be employed to greater effect.

IV. FURTHER DECREASING PROBE TIME

The optimised algorithm in [9] is highly effective at reducing the time and amount of traffic generated during the server discovery process. However, there are a number of ways of improving the process. These include decreasing the number of calibration probes and alternative sub-clustering algorithms. In this section we will explore both of these alternatives.

A. Alternative Choices in the Number of Calibration Probes

The number of samples taken from each AS cluster for calibration was nominally based on the square root of the number of game servers within that AS. We investigated this seemingly arbitrary choice through the exploration of different functions used to define N_{sample} and via a second – alternative – method in selecting N_{sample}.

We examined the sensitivity to accurate reordering of clusters on the number of sample probes. We investigated changing the scaling factor for N_{sample} from the original algorithm whilst retaining the original square root function.

A summary of the results from the client in Japan can be found in Table I. Even with as few as 50% of the original number of sampled probes, the optimised server discovery algorithm remains 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} based on prioritising the sampling of larger systems was also investigated. The rationale behind this decision is that servers within smaller AS clusters are more likely to be closer together (and share similar RTTs from the player). We propose probing a single server for these smaller clusters while retaining the original calculation for N_{sample} for larger clusters. The sampled data showed a clear delineation between clusters where < 100 servers is chosen as the size of a small cluster.

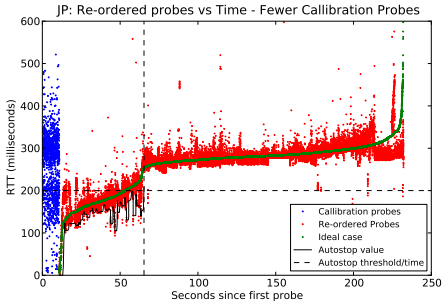

Fig. 7. Reordering issues with fewer calibration probes, CS:S client in Japan

A similar trend emerges for the Japanese client as seen in Table II. Savings in the number of 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 results is the effect on reordering. Even though the percentage of servers found with desirable latencies remained highly consistent, the reordering was less accurate as the number of calibration probes was reduced. By sampling less data in each cluster, we are less likely to locate anomalous clusters that need to be further subdivided. This is illustrated in Figure 7. It can be seen that, especially towards the end of the complete discovery process, that some clusters of servers are sub-optimally reordered.

B. An Alternative Sub-Clustering Algorithm

We attempted to improve the sub-clustering algorithm by increasing its flexibility and better selecting which servers to probe during the callibration phase. Armitage's [9] sub-clustering algorithm involves grouping autonomous systems into /16 networks if the 20^{th} and 80^{th} percentile measurements for the cluster differ by more than 40ms. We explore the impact of using a dynamic prefix length based on the number of servers selected to probe from that cluster.

The rationale behind our approach is that given a desired number of sample probes, sampling based on a fixed prefix length may lead to some subnets being probed multiple times

1) Retrieve all game servers and AS numbers
2) Cluster servers by AS number
3) For each AS cluster,
   a) N_{sample} = Number of servers to probe in cluster
   b) Select a prefix length P_{actual} such that N_{actual} = the number of subnets resulting from dividing the AS using P_{actual} and $N_{actual} \leq N_{sample}$
   c) Probe N_{actual} servers from the cluster, one randomly selected from each subnet using prefix P_{actual}
   d) $RTT_{cluster}$ = median RTT of the N_{actual} sampled servers in the cluster
   e) If the 20^{th} and 80^{th} percentile measurements of all the sampled servers differ by more than 40ms:
      i) Split the members of the cluster into new clusters – 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.

Algorithm 1: Alternate Calibration/Probing Algorithm

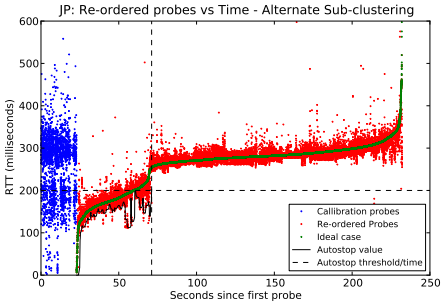

Fig. 8. Alternative sub-clustering algorithm, CS:S client in Japan

while other subnets may not be probed at all. Choosing a prefix based on the desired number of probes ensures that servers from as many possible distinct clusters in the address space are sampled.

Algorithm 1 describes our proposal. Longer prefix lengths result in an AS being divided into more, smaller, subnets. We use the nominated number of sample probes, N_{sample}, to divide the AS into N_{sample} or less subnets.

Figure 8 illustrates the effect that this algorithm has on the server discovery process for the client in Japan where $N_{sample} = \sqrt{N_{cluster}}$. In contrast to Figure 5, we see a more consistent reordering of probes – groups of servers which should have been probed later in the process are relocated correctly. A similar outcome was witnessed for the client in the UK.

We then combined the alternative sub-clustering algorithm with the decreased calibration probe count. Results for both the Japanese client and the UK client can be found in Tables III and IV respectively.

TABLE III
ILLUSTRATION OF COMBINED OPTIMISATIONS, CS:S CLIENT IN JAPAN

Algorithm	Calibration Probes	Autostop (time and % all probes)		% all probes $< RTT_{stop}$ found
Unmodified	3,185	74.5s	32.1%	100%
Modified	1,362	65.1s	28.1%	99.6%
1 probe per AS	1,176	55.8s	24.0%	84.8%

TABLE IV
ILLUSTRATION OF COMBINED OPTIMISATIONS, CS:S CLIENT IN THE UK

Algorithm	Calibration Probes	Autostop (time and % all probes)		% all probes $< RTT_{stop}$ found
Unmodified	3,136	227.7s	97.9%	100%
Modified	1,366	226.2s	97.3%	100%
1 probe per AS	1,176	226.2s	97.3%	100%

Unmodified represents the results from using the original optimised server discovery algorithm as described by Armitage [9]. The results from using a combination of the alternative sub-clustering algorithm (Algorithm 1) and a reduced number of calibration probes are referred to as **modified**. In this instance we used the values ($N_{sample} = \sqrt{N_{cluster}/8}$ for clusters with ≥ 100 servers, and $N_{sample} = 1$ otherwise). Finally, results from the extreme case of randomly probing one server in each AS cluster are reproduced for comparison.

For the Japanese client, we are now performing only 43% of the originally required calibration probes, resulting in a further reduction 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. Compared to the extreme case, we have achieved good results with only 15% more probes than absolutely required.

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 is due to the majority of servers being under the *playable* RTT threshold. Even so, the modified algorithm still managed to find all *playable* servers using fewer calibration probes.

Figures 5 and 9 contrast the original and modified algorithm's performance for the Japanese client while Figures 6

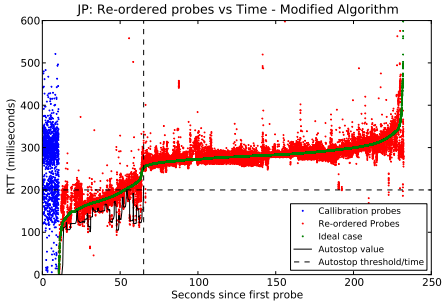

Fig. 9. Modified discovery algorithm, CS:S client in Japan

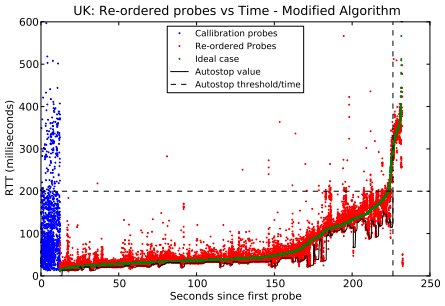

Fig. 10. Modified discovery algorithm, CS:S client in the UK

and 10 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.

Figure 10 also shows that significant time savings could be achieved using a lower *playable* server threshold (say 100ms). This may make sense where a client is close to many servers.

The decreased number of probes required to locate *playable* servers has a beneficial impact on all network users. This is achieved in one of two ways:

1) If we maintain the existing probe rate, server discovery completes more quickly and the period during which bandwidth is *stolen* [8] is minimised
2) The probe rate can be reduced and discovery can still complete in a reasonable time frame. The reduced number of non-congestion reactive packets has less of an impact on competing network traffic [8]

V. Conclusion

The network traffic generated by online FPS games can be disruptive to wireless networks. This is not limited to traffic generated while playing and, in particular, can include traffic generated during the server discovery phase. The current approach of sequentially probing each game server can take up to four minutes and generate large bursts of network traffic.

This can have detrimental effects to other users of the wireless network, an approach that can decrease the required number of probes would minimise network disruption. Armitage [9] previously proposed an algorithm where game servers are clustered based on their Autonomous System (AS) number. The clusters are then sampled, and probed in order of increasing latency. The process can be terminated early when latencies go beyond a nominated value.

We modify this algorithm to further reduce the number of calibration probes and use a different sub-clustering regimes to improve reordering. Our results show that the server discovery phase for our data set can in some instances be reduced from four minutes to one minute – a further improvement of 13% on the results achieved by Armitage [9]. Further, the ordering of probes is improved such that a user may choose to manually terminate the search earlier.

In the worst case scenario where many servers are *close* to the client, the discovery process is no longer than the existing approach and the improved sorting of probes may allow for the use of lower thresholds or early termination by the user.

References

[1] Y. Liu, J. Wang, M. Kwok, J. Diamond, and M. Toulouse, "Capability of IEEE 802.11g Networks in Supporting Multi-player Online Games," in *2nd IEEE International Workshop on Networking Issues in Multimedia Entertainment (NIME 2006)*, Las Vegas, NV, January 2006, pp. 1193–1198. [Online]. Available: http://www.cs.umanitoba.ca/~yliu/MYPUB/nime2006.pdf

[2] B. Carrig, D. Denieffe, and J. Murphy, "Supporting First Person Shooter Games in Wireless Local Area Networks," in *IEEE 18th International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC'07)*, Athens, September 2007. [Online]. Available: http://dx.doi.org/10.1109/PIMRC.2007.4394387

[3] G. Armitage, M. Claypool, and P. Branch, *Networking and Online Games - Understanding and Engineering Multiplayer Internet Games*. UK: John Wiley & Sons, 2006.

[4] G. Armitage, "Discovering First Person Shooter Game Servers Online - Techniques and Challenges," *International Journal of Advanced Media and Communication (IJAMC)*, vol. 2, no. 4, pp. 402–414, 2008. [Online]. Available: http://www.inderscience.com/filter.php?aid=22223

[5] T. Henderson, "Observations on Game Server Discovery Mechanisms," in *NetGames '02: Proceedings of the 1st workshop on Network and system support for games*, Braunschweig, Germany, 2002, pp. 47–52. [Online]. Available: http://doi.acm.org/10.1145/566500.566507

[6] M. Claypool, "Network characteristics for server selection in online games," in *ACM/SPIE Multimedia Computing and Networking (MMCN)*, San Jose, California, January 2008. [Online]. Available: http://dx.doi.org/10.1117/12.775138

[7] W. chang Feng and W. chi Feng, "On the Geographic Distribution of On-line Game Servers and Players," in *NetGames '03: Proceedings of the 2nd workshop on Network and system support for games*, Redwood City, California, 2003, pp. 173–179. [Online]. Available: http://doi.acm.org/10.1145/963900.963916

[8] T. Nguyen and G. Armitage, "Quantitative Assessment of IP Service Quality in 802.11b Networks and DOCSIS networks," in *Australian Telecommunications Networks & Applications Conference 2004 (ATNAC 2004)*, Sydney, Australia, 8-10 December 2004, pp. 121–128. [Online]. Available: http://caia.swin.edu.au/pubs/ATNAC04/nguyen-t-armitage-ATNAC2004.pdf

[9] G. Armitage, "Optimising Online FPS Game Server Discovery through Clustering Servers by Origin Autonomous System," in *ACM NOSSDAV 2008*, Braunschweig, Germany, 28-30 May 2008, pp. 3–8. [Online]. Available: http://dx.doi.org/10.1145/1496046.1496048

[10] Y. Liu, J. Wang, M. Kwok, J. Diamond, and M. Toulouse, "FPS Game Performance in Wi-Fi Networks," in *4th International Game Design and Technology Workshop and Conference (GDTW2006)*, UK, November 2006, pp. 41–49. [Online]. Available: http://www.cs.umanitoba.ca/~yliu/MYPUB/gdtw2006.pdf

[11] G. Armitage, "Client-side Adaptive Search Optimisation for Online Game Server Discovery," in *IFIP/TC6 NETWORKING 2008*, Singapore, 5-9 May 2008, pp. 494–505. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-79549-0_43

[12] "Master server query protocol," July 2009, http://developer.valvesoftware.com/wiki/Master_Server_Query_Protocol.