Project: MLC SS04

Better Game Server Selection

A Major Qualifying Project Report:

submitted to the Faculty

of the

WORCESTER POLYTECHNIC INSTITUTE

in partial fulfillment of the requirements for the

Degree of Bachelor of Science

by

_____ _____

Steven Gargolinski **Christopher St. Pierre**

Approved:

_____

Professor Mark L. Claypool, Advisor

Abstract
As the gaming industry grows, so does network gaming as a whole. In recent years, as network gaming has evolved, server browsing has not. Today's simple static ping times are not enough to accurately choose a game server to play on. Our approach provides a windowed average of ping times that better serves the player, in addition to providing more efficient ways to choose a game server for multiple players.

Table of Contents

1. Introduction

In recent years, the video game and interactive entertainment industries have grown up from their largely underground roots and established themselves as respectable and profitable media outlets. No longer visually abstract or requiring a major suspension of disbelief, video games have now appeal to a much wider audience, and are currently more popular now than they have ever been before.

Late 2004 saw the release of many highly-anticipated, blockbuster video game titles including Doom 3, Half Life 2, Grand Theft Auto: San Andreas, and Halo 2. Titles such as these led the way as the video game industry pulled in just under $10 billion of revenue in 2004 [SALES2]. Halo 2 for Microsoft's Xbox console sold nearly 2.5 million copies in one day of sales, totaling roughly $125 million in sales. By comparison, Hollywood's biggest opening weekend of the year belonged to "Spiderman", which brought in about $114 million [SALES].

Online gaming is also more popular now than ever. Three of the four best-selling games mentioned above have significant online components. At any given time, hundreds of thousands of players are competing with each other online. According to the NDP group, World of Warcraft (the latest massively multiplayer online game) is currently the best selling PC game, selling 240,000 copies in the first 24 hours of sales [WOW].

Many of these games require the player to select which server they are going to play on. Game server browsers have not evolved much during the rise of multiplayer gaming with the vast majority still simply reporting a list of servers with a bit of minimal

information about each one. The goal of this project is to improve the player's server selection process.

The first problem addressed in this project is the inability to select a server for multiple players in geographically separate locations effectively. There currently exists no good method for two players in two separate locations to choose a multiplayer game server which will perform well for both of them. The only way to currently accomplish this is through a manual trial-and-error type approach, and requires some sort of external communication. Players view their server browsers independently and one player suggests a server (over telephone, instant message, or some other external method of communication). The rest of the players locate this server in their own browser, decide if it is acceptable, and report back to the group. If each player agrees that this server is acceptable then the game can be joined. If not, it is necessary to repeat the process.

This process is obviously cumbersome, slow, and ineffective. The amount of time it takes even a small set of players to search a decent-sized subset of the available servers makes this method virtually unusable. To choose the best server, a set of players would have to iterate through each available server, comparing their ping times for each one, and only making a final decision after examining ever available server from each players point of view.

The solution presented by this MQP largely automates the process described above. Multiple clients connect to one process which acts as a host. The host specifies game options and server requirements. For example, the host might run a search for a Quake III server playable for three players, which is running a Deathmatch game on the "Apocalypse Void" map. Each individual client sends information about every potential

server in the world to the host. The host analyzes this gathered data and makes a decision on which server has the most potential to provide a low-latency gaming experience to this set of clients and sends the command back to the client required to launch the game on the chosen server.

At the worst, this utility will arrive at the same answer as a group of players who decide to examine each and every server manually. At the best, this tool will utilize an optimal server-selection algorithm, choosing not only the best possible server, but choosing it in a matter of seconds, not hours.

The second problem examined is the inability of a single ping value to reflect the current load of a server. Currently, modern server browsers only ping each server in their list one time and display this value to the user to as a measure of its playability. This small sample set is unreliable. A network or server system anomaly could cause an erroneous ping time which could makes the server appear unplayable when in fact the server would perform very well. Additionally, it may be possible for a server to appear good, but be unusable as a game server.

If a server captures the amount of time it takes to ping itself, the value returned indicates the amount of time necessary to return a ping packet with zero network delay. This value is an indication of how loaded the CPU of the host computer is. This value will be referred to as a server's "local load" throughout the rest of this paper.

If there are many users currently connected to a server and the CPU load is high, then this will also be reflected in the local load. After some preliminary investigation, it was determined that this local load is also reflected in the ping value of any server as queried from a remote client.

Utilizing this information, two alternate server browser implementations were produced to better equip players with the information necessary to select an optimal server.

The first implementation relies on servers to run an external process which continually gathers local load information, calculating statistics and listening on a well known port in order to report this information back to the server browser which in turn presents it to the user.

The second implementation does not require any sort of change on the server end of the problem. Instead, it gathers ping values from each server in the normal fashion, but takes it one step further and stores the ping values locally. The more times a user refreshes the server list, the more data they are able to base their decision on. Instead of presenting the user with a single ping value, the server browser instead presents the user will a history of ping values.

Each of these two alternate server browser strategies will allow the user to make a more accurate server selection. This MQP is organized in four separate sections: history, tools, the problem with lag and playability, and the problem with server selection, followed by our conclusions and summaries.

2. Background

2.1 History

In 1958 William A. Hinginbotham, working at the Brookhaven National Laboratory used an oscilloscope to simulate a virtual game of tennis. This crude creation utilized an overhead view, allowing two players to compete against each other in an attempt so sneak the ball past the paddle of their opponent. Hinginbotham called this game "Tennis For Two" [PONG].

In 1972 an electrical engineer named Nolan Bushnell formed Atari, a company which at the time was dedicated to programming and producing arcade games. One of the earliest games they produced was known as *Pong*, an arcade friendly version of Hinginbotham's *Tennis For Two. Pong* became the first commercially successful video game.

In the years after *Pong*'s release, video games were not extremely popular. For many years the video game industry was overshadowed by movies and television. They brought in less money, appealed to a smaller percentage of consumers, and thus were given less shelf space and exposure.

In the early to mid-nineties, this began to change. Computer power was increasing rapidly, allowing video games to produce more realistic graphics and sound. Players were no longer forced to go to great lengths to suspend their disbelief. Instead of controlling a square moving slowly around on a four color screen, they were able to zip around a 256-color environment, heightening the overall experience of a more realistic, lush, virtual world.

Nintendo released its *Nintendo Entertainment System* home video game console in 1985, and is widely credited for saving the United States video game industry which had suffered a crash in 1983. *Super Mario Brothers 3* sold over 17 million copies, and was featured in a Hollywood movie [SMB3].

The shareware version of Id Software's *Doom* was installed on more computers than Windows [MOD]. Intel, along with colleges across the country implemented specific anti-*Doom* policies in an attempt to unclog their computer networks and restore employee productivity. Fast-paced, violent, with bleeding-edge graphics, *Doom* was surely a sign for future advancements in the video game world. *Doom* was the first game to include the now very popular game mode known as "Deathmatch", in which players compete against each other across a computer network, putting their skills to the test an attempt to earn more "frags" or kills than their opponent.

In the time since the mid nineties, the video game industry has increased its foothold in the entertainment market even more. Late 2004 saw the release of many highly-anticipated, blockbuster video game titles including Doom 3, Half Life 2, Grand Theft Auto: San Andreas, and Halo 2. Titles such as these led the way as the video game industry pulled in over $10 billion of revenue in 2004 [SALES2].

2.2 Multiplayer Network Gaming

Multiplayer network gaming really took off in 1996 with the release of Id Software's *Quake*. Games at the time were generally played by several players over a LAN, or by two players connected to each other through the phone lines by way of their modem.

Quake featured a method allowing players to compete against each other over the Internet. In the Doom days, players were forced to coordinate times and places in order to find a game. These problems were solved by the inclusion of Internet-based multiplayer gaming in *Quake*. Persistent servers were set up and ran 24 hours a day. Players from all over the world could connect to these servers at any time of day or night, always able to find a game.

These servers acted as hosts, and would be in charge of executing the server application for the game. Players would connect as clients, logging in for a game session to fight with or fight against other players. The player's input was sent to the server, which would keep track of the state of the world. Information about the world would be periodically transmitted back to the players, updating their view of the world to match the one currently running on the host machine.

This paradigm was not without its problems. It worked well on a LAN with high-bandwidth connections capable of sending transmissions amongst the players at a rapid rate. Unfortunately not all connections were capable of such high transmission speeds; most players at the time were connecting to the internet through relatively low-speed dial-up modems.

In 1996, standard-model modems transmitted at 14,400 bps. This led to high ping values, indicating that packets of data took too long to travel across the internet to their destination. Ever since this time, players interested in online games have been interested in maximizing network performance, minimizing their ping time. Low ping times lead to smoother gameplay, which leads to more control responsiveness, leading to more accurate movements and hopefully a higher score. The search for lower internet ping times had begun.

Luckily, Id Software realized the problem and quickly responded by taking a big step forward in accounting for network latency. *QuakeWorld* was released later in 1996 as a free add-on to *Quake*. *QuakeWorld* included a number of updates in addition to rewritten network code. It allowed players to tweak several parameters in order to minimize the effect of their slow internet connection on gameplay. These tweaks included the number of packets the server would send them each second, as well as the ability to set push latency [QWORLD]. *QuakeWorld* also implemented a technique known as client-side prediction. Clients no longer had to wait for data from the server in order to update the state of the game world. They were now able to partially predict the future game state, updating it at more regular intervals.

At this point players all over the world, with all different types of internet connections were now able to find a high-performance game of *Quake* at any time of the day. The stage was set for the inevitable rapid growth of online-gaming. The industry, the developers, the technology, and perhaps most importantly the players definitely did not disappoint.

Multiplayer gaming took on a culture all its own. Addicts played all hours of the night. Gifted players gained legendary reputations with new players always gunning to take down a champion. Players would often form up into Clans, practicing and competing with each other, spending hours together daily. Despite the amount of time involved, these relationships were often completely limited to avatar-based, online interactions. A few weeks after the release of *Quake*, there were thousands of organized Clans.

LAN parties and tournaments became more and more popular. Players would regularly pack up their machines in order to compete against each other in a localized setting, often organizing these events into tournaments. Amongst the most popular is QuakeCon which is often referred to as the Woodstock of gaming. Gamers come from all over for four days of "peace, love, and rockets".

Players would compete online for a chance to win entry into the bigger tournaments, often battling it out for generous cash prizes. Thousands of players competed for the chance to enter Id Software's Red Annihilation tournament. The winner of the deathmatch tournament took home John Carmack's turbo-charged, cherry-red Ferrari [MOD].

Multiplayer online gaming even became the full-time job of some gifted players. Dennis 'Thresh' Fong earned well over $100,000 in 1998 competing in *Quake* [MOD]. An organization known as the Cyberathlete Professional League (or CPL) was started in the late nineties with a goal of bringing in crowds of spectators to watch live deathmatch tournaments [MOD].

People are playing online games now more than ever. Nearly one hundred thousand people can be found playing *Counterstrike* at certain times of day [CSTRIKE]. A new genre of game, the Massively Multiplayer Online Role-Playing Game (or MMORPG) allows unprecedented numbers of players to inhabit a common online environment at the same time.

Online gaming used to only exist in the PC world, with game console systems were still offline for the most part. This is no longer the case. Each of the three major consoles (Sony's Playstation 2, Microsoft's Xbox, and Nintendo's Gamecube) is equipped with online capabilities.

Microsoft's online Live service has over 1 million subscribers [LIVE]. Square-Enix's MMORPG game *Final Fantasy XI* has the ability for PC users and console users to intermingle in a common world.

The upcoming generation of handheld gaming systems (Sony's Playstation Portable, and Nintendo's DS) are equipped with a wireless mode (following the IEEE 802.11 standard) for multiplayer gaming as well as the ability to play certain titles through the internet.

The rise of gaming into a major economic and social force is due in no small part to online-multiplayer capabilities. Taking into account the ever-increasing popularity of online games, along with the ever-increasing percentage of homes possessing a broadband connection, there is every indication that the importance and participation level of online games will only continue to grow as we move forward into the next generation of games.

This significant level of growth in online games indicates a need to develop not just better games, but also a better set of tools and utilities allowing players to easily find a suitable server to play on at any given time. Games evolve leaps and bounds with each subsequent generation, but the server selection paradigm has remained relatively stagnant. Moving forward, the gaming experience can be greatly enhanced through improving this server selection process, allowing players to connect to high-performance servers to game with their friends in a fast, efficient manner.

2.3 Game Server Selection

Game server selection has not evolved much in the decade or so since the first modern server browser was released. A server browser generally displays a fairly small set of information, normally providing fewer than ten ping times for the player to use to make his server decision. In many games this consists of only the server name, the map currently being played, the number of players currently connected, the maximum number of players this server will allow, and the game type (Capture the Flag, Deathmatch, King of the Hill) as well as any special modifications this server is running.

The browser will also produce a rating which attempts to quantify the playability of the server. For the most part, this rating is captured as a ping value, in milliseconds from the client to the server and back (round-trip). In an effort to increase the speed at which the possible server list is displayed, this ping value is generally taken from a measurement of a single packet.

Another popular server option is to require Punkbuster [PUNKBUSTER]. Punkbuster is a utility designed to prevent players from cheating, and runs on the player's

client. Many servers require that players have Punkbuster running before they will allow them to connect in an effort to cut down on the amount of cheating on their server. Normally a Punkbuster icon will appear next to a server if it is required.

The information described above is generally provided for most every online game. Depending on the specific game, there may be another set of information that is necessary for the player to know in order to properly select a game server. This information may include facts such as number of teams, team size, available weapon sets, or available vehicle sets.

Server browsers generally allow players to sort the server list by any piece of information, in order to emphasize the servers which meet their requirements the best. For example, a player who wants to play a game of Capture the Flag can sort the server list by game type. This player would then be able to quickly and easily view all of the potential Capture the Flag servers.

Another desirable piece of functionality in a server browser is the ability to sort servers based on ping. This is generally done in an attempt to select the server which will perform the best once the player joins the game.

Figure 1: A Typical Quake III Server Browser Screen

Figure 1 shows a typical Quake III server browser that the player would see on the screen after clicking multiplayer. It is currently sorted by ping time, and various game options have been selected. By clicking refresh, the list would be repopulated and resorted. This list is generated through querying the game's master server, requesting data on each game server currently registered with this master server. After picking a server in the list, the player could click FIGHT and join the game server.

3. Tools

3.1 QStat

QStat is a command-line program that gathers real-time statistics from Internet game servers [QSTAT], information that a game server browser can use to allow a game client to select an appropriate server. Most games that QStat supports are first-person shooters (Quake, Half-Life, etc.) and since most first person shooters support server browsing using either the Quake or Unreal protocols, QStat also works with most first-person shooters.

QStat obtains the latency information for each game server by sending an application-level ping packet over UDP to the game server. Upon receiving the ping packet, the game server responds with basic game information. The data in the ping response can be formatted in several ways, including XML and a specified RAW format. Upon receiving the ping response, QStat records the latency as the time between sending the ping and receiving the response. In the event the ping or the ping response are lost (or the server is down), QStat will repeat the ping request two additional times.

QStat also has the ability to query a master server. A master server contains a list of servers currently registered as active games. When run on a master server, QStat receives a list of all registered servers and runs the specified query on each of these servers.

For some examples, the following command obtains detailed server information for all Quake III servers registered with the master server:

```
qstat –q3m,68 master3.idsoftware.com –R –P
```

and the following command obtains basic server information from a specific Quake 3 server at port 27960:

```
qstat –q3s 216.12.96.41:27960
```

3.2 QStatExt

QStatExt is an extension to the game server utility *qstat* which provides several additional functions and enhancements. Most immediately noticeable is the graphical enhancement. QStatExt is a windows utility which wraps the command line options required to utilize *qstat* inside a familiar set of standard Windows graphical interface controls.

This tool provides the capabilities to quickly and easily gather a large amount of data about players currently connected to game servers. For example, QStatExt allows the automated collection of ping time and score of each player logged into any Quake III server every ten seconds. The tool also allows specific collection of data about each player on a specific server, allowing the quick and easy automated collection of data describing each player which logs into a single specific server.

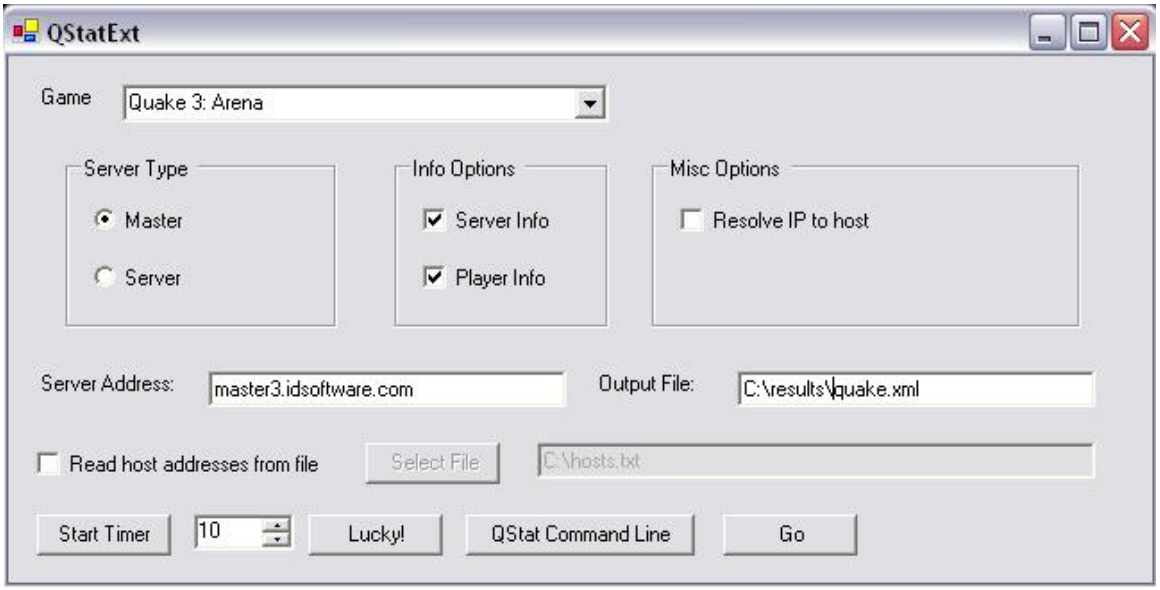

Figure 2: QStatExt Sample Screenshot

These controls allow a user to select information such as the game, server address, whether the server should be queried for game rules, and whether or not the server should be queried for player rules. It also provides a control to set the path and file name to save the collected data.

QStatExt also allows the user to specify if the player is querying a standard server or a master server. If the program is querying a standard server, it is only interested in that single server. In the case of a master server, the player first receives a list of servers which are registered with the master server and then take a look at each of these servers one at a time.

Either way, the resulting information is formatted and output into a known-format as an XML file. At this point there is one more round of processing performed on the data. The XML is read into internal data structures and processed, sorted to separate the

information into different lists, one for the player info and game rules for each individual server.

This information is output into two separate CSV (comma separated value) files. The first CVS file is basically a list of irrelevant servers. QStatExp dumps out a list of servers which are currently down, meaning that the player's ping attempt timed out (ping time of 999 is returned) into this first CSV file.

The second CVS file contains a list of the relevant information gathered by our server query. The exact information output can changed based on which game is currently selected. The player information returned from the server is different based on the game's specific protocol. The information available could very well change from game to game. Games utilizing the same engine will generally return the same set of data for each player. For example, a certain protocol might return each player's current score and ping time, another protocol might return score, ping time, as well as the team number the player is on.

Assume for example that we are querying the master Quake 3: Arena server. The first step is to query the master server, which will return a list of individual game servers. The next step is to query each of these servers, capturing the player information and server rules in XML format. Once the XML has been constructed, the next step is to read the XML into internal, sorted data structures. At this point, the servers are iterated through and the ones which timed out are output into a CSV file. The next step is to create a CSV of the relevant information which was not output in the previous step. For this example the game is Quake III, so an assumption will be made that the interested

information is player name, the server the player is competing on, this player's ping, and the player's current score.

QStatExt extracts all of this information and outputs the information in CSV format. This data can easily be read into Excel, Gnumeric, or another spreadsheet program and analyzed in order to extract averages, statistics, or any other miscellaneous desired data trends.

QStatExt also provides timer functionality in order to streamline the data collection process. A spinner control specifies a number of seconds, allowing the user to specify the granularity of the timer. When the timer expires, QStatExt will gather data based on the options currently specified. The timer will then reset and wait for the next iteration of the cycle.

QStat will output files with a number appended on the end which increments each time through the cycle. For example, assume that the output file names chosen by the client are C:\quake.xml and C:\quake.csv. The first time data is gathered, C:\quake1.xml and C:\quake1.cvs are generated, the second time C:\quake2.xml and C:\quake2.csv will be generated. Future iterations will follow this naming scheme, continuing until the timer is stopped or the program is closed.

3.3 Server Browser Extension

A single ping value is not an accurate determination of a server's performance, therefore it is necessary to keep a running average or ping history in order to allow the user to make a more accurate, informed, server selection.

3.3.1 Information Stored on the Server

The "Information Stored on the Server" method requires two pieces to work correctly. It requires a process to be run on any number of game servers, as well as an alternate server browser which will present the information to the player.

The process which runs on the server has two basic functions, which are implemented as two separate threads. First of all, the server browser is responsible for keeping track of the server's local ping time. This involves running qstat on a configurable timer and keeping track of the results. In this case, qstat is set up to record the local ping of the server running the process.

This value represents the local ping, or the server's ping time to itself, which negates any network delay that would normally be involved in a ping time. Our research has shown that this local ping time captures local load variance very well.

This value is stored locally, added to a configurable variable-sized array of ping data. Analysis is performed on this data, calculating a number of different metrics such as the average, standard deviation, minimum value, maximum value, and ninetieth percentile value.

The second function of the server process involves listening for a client request. The program sits on a well-known, advertised port listening for users. When a client

connects, the server transmits a block of data to the client, reporting all of the data metrics that we calculated in the previous step.

The second part of this solution is on the client's end in the form of a customized game server browser. This is implemented in a way so that even if no servers are running our utility described above, this program will behave basically exactly like a normal game server browser.

The utility uses qstat to query the game's master server in order to obtain a list of each of the individual servers as well as all of the basic information regarding these servers. This is the normal process carried out by an ordinary server browser. At this point an ordinary server browser finishes it's responsibilities by presenting this gathered data to the user.

Instead of presenting the information, this custom server browser instead gathers some additional data. Each server's IP address is known, so this browser utilizes this address along with an advertised port to connect to the server process described above. If the server is not running the custom process, then this connection will fail and the player will not be able to obtain additional information on this server.

Assuming that the client process is able to connect to the server process, the data gathered on the server will be transmitted to the client. This process is repeated for each server the player will consider.

At this point, data will be displayed to the user in basically the same fashion as a normal server browser, except in this case the additional statistics gathered by the server process will also be displayed. In the case of a server where the client was not able to

obtain this additional information, the server browser will default to the standard information display.

3.3.2 Information NOT Stored on the Server

The second type of server browser tool does not require the actual game servers to run any sort of additional programs as is necessary in the first alternate server browser implementation. In this version the client is responsible for gathering and analyzing the server data.

This server browser begins by utilizing qstat to gather a list of each game server registered with the master server. The browser stores this data in an internal structure (a map), which maps the server address to a list of data describing ping information for a single server at a given point of time.

This data is analyzed in the same fashion as we did above, minimum, maximum, ninetieth percentile value, etc. At this point the program only has one ping value for each server, so the calculations are trivial. This data is formatted and presented to the user.

At this point the user has two options. If they choose, they can browse through the server data and pick a game to play. Or they can choose to refine the data currently being presented to them. This involves running qstat once more and gathering the data on each server registered to the master server. The browser must be run manually and outside of the game.

At this point, the server browser adds this new data to the map created the first time qstat was run. Now the program has a larger set of data and is able to produce slightly more meaningful statistics. The server browser calculates a new set of statistics,

most likely based on only two data points per server, and presents this information to the user.

This iterative approach allows for many different data gathering methods. The program can be configured to update the server list at a timed interval, each iteration will provide a more accurate picture of the probable server performance. This method makes a trade off, normally taking a bit of extra time up front in order to obtain several iterations of server data in the hopes of finding a server with good performance more quickly and efficiently. Future work in this area could provide a recommendation as to a number of runs which would constitute a legitimate approximation of the current server performance.

This server data could even be stored as a data file on disk for future analysis. The next time the server browser is launched, instead of starting over from scratch with zero data points, it can read in a set of ping data history from the previous times the tool has been run.

3.4 MUSST

3.4.1 Program Flow

When using the Multi-Location Server Selection Tool there are two basic types of users. The first type is a “host”, responsible for setting up the game. The second type is a “client”, responsible only for connecting to the host. There is always exactly one host, who may or may not also be a client. Any number of players may connect to a single host.

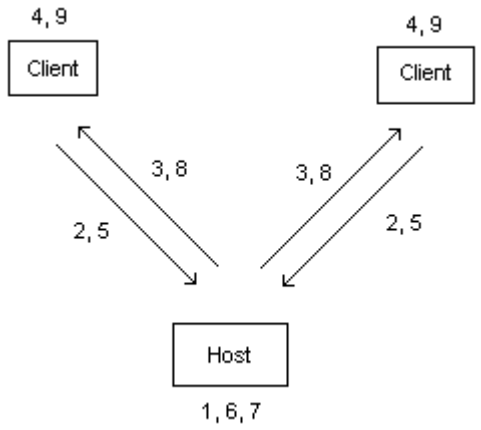

Figure 3: Architecture and Interaction for Game Server Selection for Multiple Players

The host first selects a variety of program options such as the game, number of players, desired maps, or a specified game type (Deathmatch, Capture the Flag, etc.). The host then goes into a listening state, waiting for the specified number of clients to connect (step 1 in Figure 3).

Each client is now responsible for connecting to the host (step 2). For this to take place, the clients only need to know the IP address of the host. No other options are specified on the client side.

The host remains in its listening state until the number of players that want to play together have connected. At this point, it uses the options specified in order to build a *qstat* command line which will allow the clients to gather the necessary data from the game's master server. The server then transmits this command line data to each of the clients (step 3).

Upon receiving the command line options sent by the host, each client executes *qstat* and collects this data describing all relevant game servers in XML format (step 4).

When the client finishes gathering the data, it transfers this information back to the host (step 5).

The host receives this XML data until it has collected one set of data from each connected client. The host parses the data files from each client, reading the information into internal data structures (step 6).

This data is then run through a series of algorithms in order to make an informed decision about which server is the optimal choice (step 7). This server selection is then sent back to each client along with the proper command required to launch the chosen game and connect to the indicated server (step 8).

Finally, each client launches the specified game and joins the server located at the address and port number decided upon by the host (step 9).

MUSST also contains the ability to re-evaluate old sets of data. This is useful to test different algorithm outputs on the same set of data. This feature also allows the ability to re-evaluate old data without the burden of requiring a number of clients to connect to the host.

This feature is also useful to test the server selection mechanism for various subsets of the set of player data. For example, assume that MUSST decides on an optimal server for the player set {1, 2, 3, 4}. The ability to re-evaluate old sets of data would allow us to compare this server selection to the one chosen for the sets {1}, {2}, {3}, {4}, {1, 2}, {2, 4}, {1, 2, 3}, {1, 2, 3}, etc.

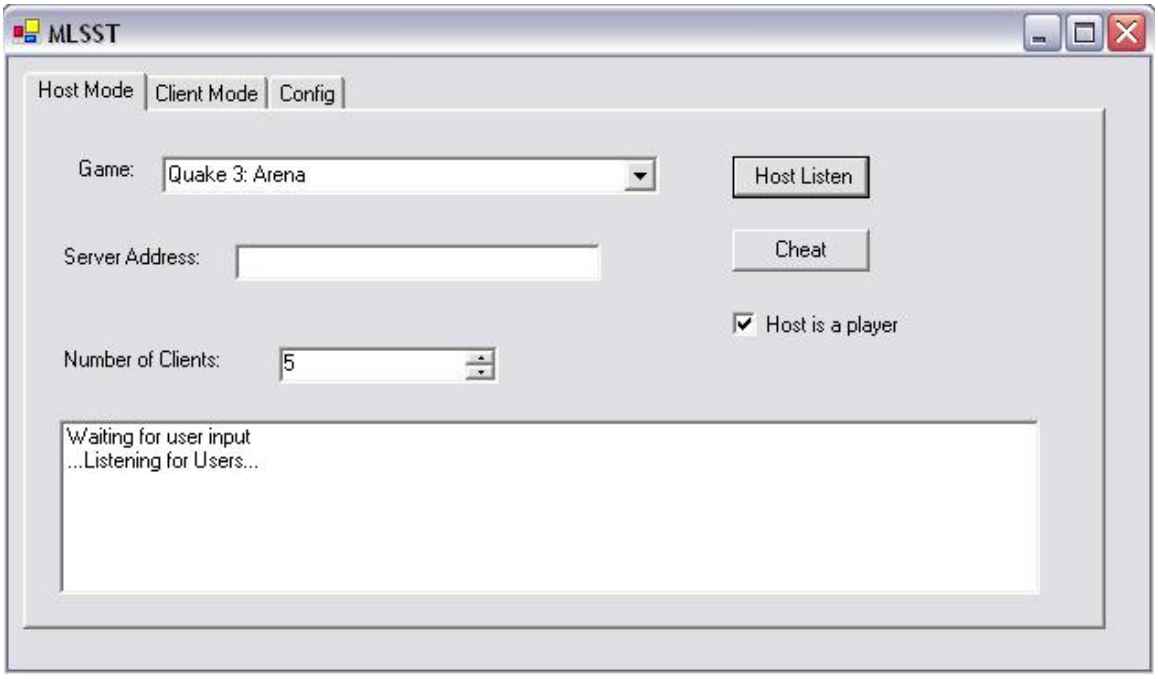

Figure 4: MUSST Application Host Tab

Figure 4 depicts the interface presented to the user who is acting as the host for this session. There is a pull-down menu listing several different games; "Quake III: Arena" is currently chosen. The number of clients expected to connect to this session is set to 5, changed by utilizing the spinner control on the layout.

The checkbox marked "Host is a Player" indicates that the user running in host mode is also going to connect to the game. QStat information will be gathered and taken into account for the host the same as each of the clients.

The "Host Listen" button is pressed once all of the options have been set. MUSST will go into its listening state, waiting for the specified number of clients to connect. The "Cheat" button is used to process existing data without requiring actual clients to connect. It reads in a number of data files specified by the "Number of Clients" and chooses a server. This feature is provided for re-processing of old player data,

allowing sets of data to be analyzed quickly without the need for physical connections to be made. Utilizing this method, algorithms may be tweaked and re-written and the results compared against each other in order to move closer to an optimal selection algorithm.

The listbox at the bottom of the control is used as a feedback mechanism. Throughout the life of this MUSST use-case it will report various pieces of information to the user.

The “Server Address” edit box is currently unused. It was added with possible future extensions in mind which would allow the host to specify an alternate master server address.

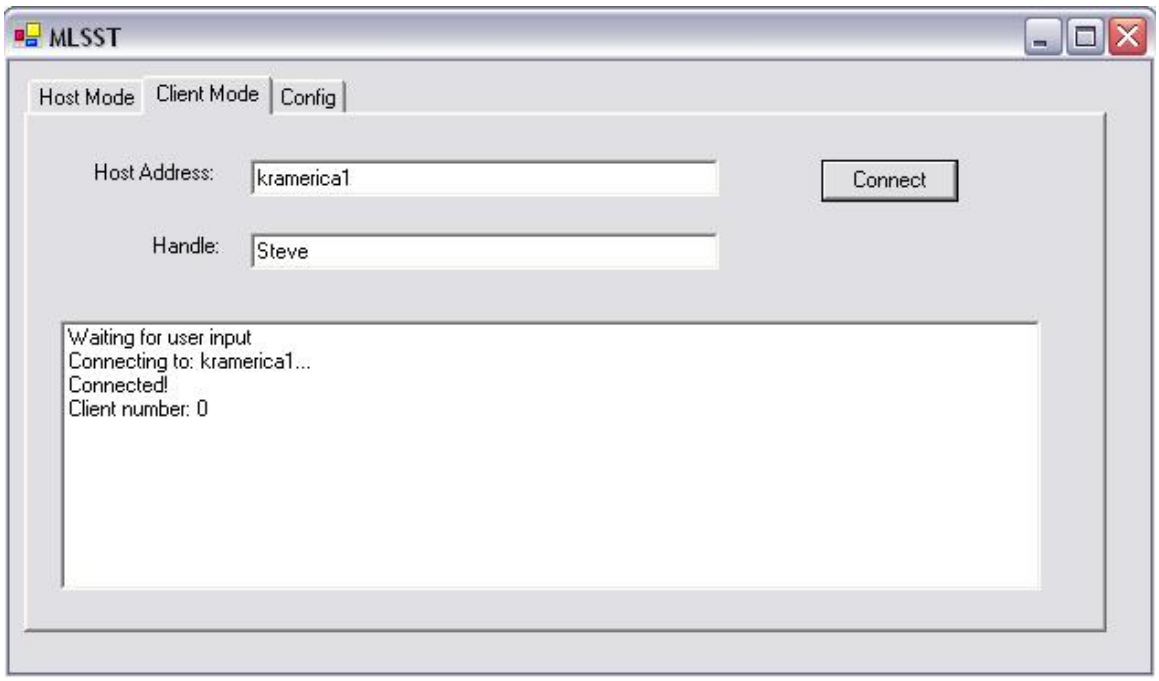

Figure 5: MUSST Application Client Tab

Figure 5 depicts the MUSST interface presented to each user acting as a Client. This layout is simpler than the host version. Only two pieces of information need to be

supplied by the client. The first is the server address of the user running MUSST in host mode. Currently this is set to “kramerica1”, the name of a local machine on the network.

The second (optional) piece of information is the player’s handle. This will appear in the status updates provided to the host, indicating that “Player connected!” or “Steve connected!”. This information is not required and only serves as a way for the host to keep track of which players have successfully connected.

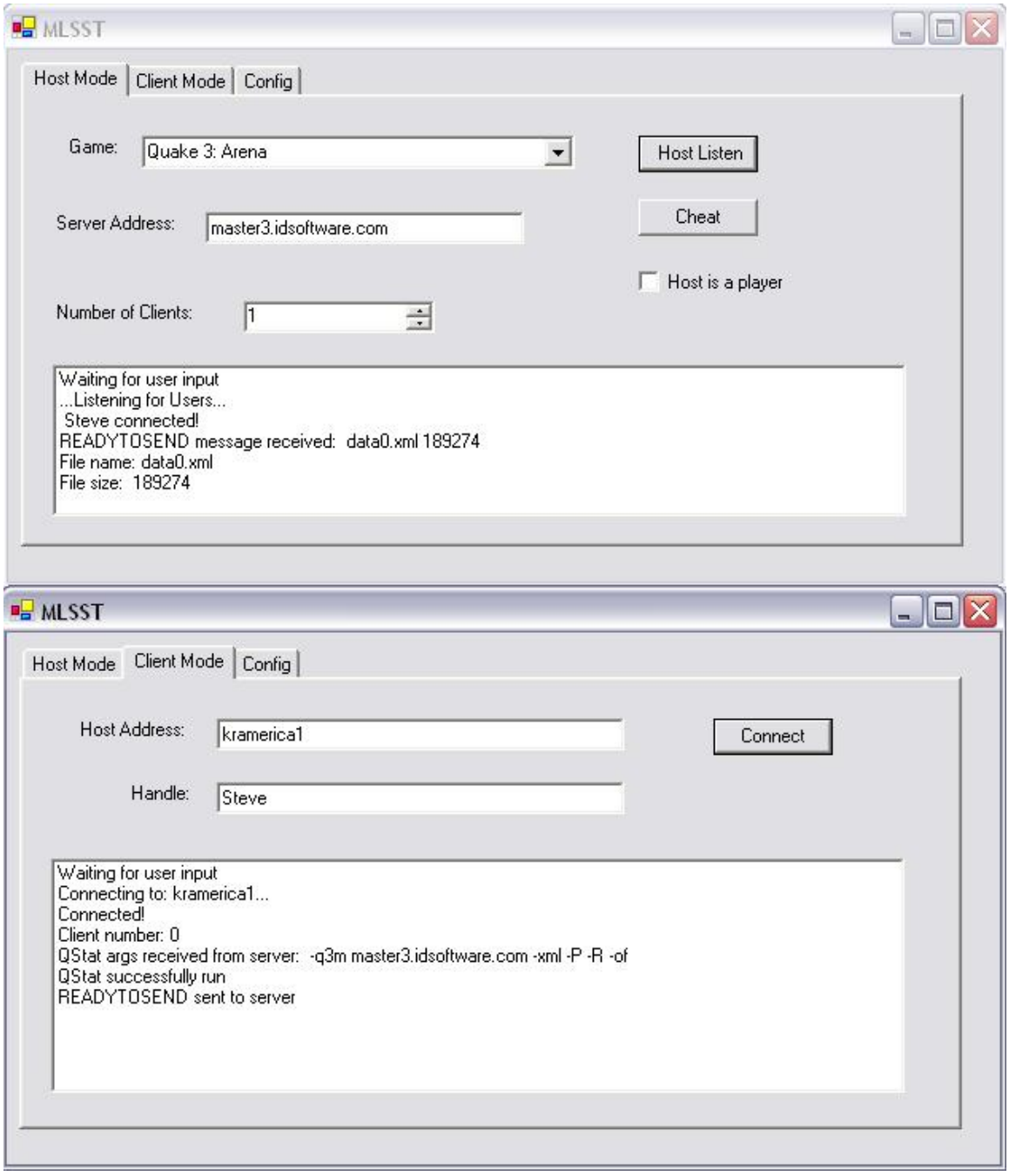

Figure 6: MUSST Application Client and Server Interactions

Figure 6 depicts the interaction between a host and client instance of MUSST. In this figure, the following steps have taken place: The host selected the game, specified

the number of clients, and pressed the “Host Listen” button in order to enter the listening state.

The client specified a handle “Steve” and also entered a host address “kramerica1” (currently on the local network, or else an IP address would have been specified). The client then pressed the “Connect” button and established a connection with the host computer.

The host accepted the client’s connection (“Steve connected!”), and built a QStat command line argument to send back to the client. The client received this command line (“-q3m master3.idsoftware.com –xml –P –R –of”) and executed QStat, specifying these command line options.

The client then transmitted its data file to the server (“File name: data0.xml”, “File size: 189274”) as the last action which has taken place in this use-case. Beyond this, the host would read in the data, select an appropriate server, and transfer this server information back to the client.

3.4.2 Selection Process

As mentioned above, the host runs the set of client data through a series of algorithms in order to choose the best server for this particular set of clients. There are two different types of algorithms used here.

The first type of algorithm is responsible for culling the data by identifying and removing any trouble servers. This process insures that the server selected comes from a valid set of servers.

For example, one algorithm makes sure that only servers in which each client has sent information about are considered. If one client fails to report a ping for a certain server, this algorithm is responsible for removing the server in question from the set of possibilities.

Another algorithm scans through the possible servers and removes the ones which do not have enough empty player slots to host the number of clients currently looking for a game. For example, if we are currently searching for a server to host 8 players, a server with a maximum player value set to 6 must be discarded as well as a server with 16 maximum players and 10 players currently connected.

Finally, the servers are scanned and the list pruned based on the options initially selected by the host. For example, at this point all non-Capture the Flag servers can be discarded or all non-Deathmatch servers. The algorithm can disregard all servers not currently running a specified map.

At this point each server in the list has been trimmed down to a smaller set. Each entry meets the requirements specified by the host at the beginning of this process.

The second type of algorithm used is responsible for decided which server from the trimmed list is the optimal solution.

The most straightforward solution is to select the server with the lowest average ping time amongst the clients. This solution may not be such a bad way to choose, but there are definitely numerous situations under which this algorithm would make a poor decision.

	Server A	Server B	Server C
Player 1	24	74	95
Player 2	17	62	89
Player 3	41	100	92
Player 4	35	51	88
Player 5	18	84	96
Player 6	27	44	87
Player 7	30	122	93
Player 8	272	71	94
Average	58.0	76.0	91.8
Standard Deviation	86.8	25.7	3.4

Table 1: Sample Ping Data for Three Players on Three Servers

For the following examples, reference Table 1. Assume there are eight clients connecting and their ping times for Server A are as follows: 24, 17, 41, 35, 18, 27, 30, 272. The average of these pings is 58. This is certainly a playable ping time as an average, but the client who had a ping time of 272 is going to have an extremely difficult time playing the game.

There are multiple ways to deal with a situation like this. For example the algorithm above could be extended to include a threshold, and only choose a server if each client's ping was within this specified threshold.

Assume that we have chosen a maximum threshold of 150 ping, meaning that we will not select any server which returns a ping greater than 150 for any of the clients. Under these conditions, Server A will be removed from consideration. We may instead choose Server B, which reports client ping times of 74, 62, 100, 51, 84, 44, 122, and 71. The average of these values is 76, which is slightly higher than the value for Server A. However, Server B becomes our choice because all of the client pings are less than 150. Using this algorithm we have selected a server which will most likely be playable for all eight players connecting.

Another possible solution would be to choose the server which has the smallest deviation amongst the client pings. Imagine a Server C with client ping times of 95, 89, 92, 88, 96, 87, 93, and 94. This average client ping for Server C calculates out to 91.8 which is higher than the value for Server A or B (58 and 76 respectfully). The average ping for Server C may be higher, but the standard deviation is much lower, 3.4 compared to 86.8 for Server A and 25.7 for Server B.

The best algorithm is certainly an area for future work. However, most reasonable algorithms will perform significantly better than any method of manual selection.

4. Procedure for CPU Load Experiments

4.1 Motivation

Using the common gamer knowledge that picking a server according the ping times does not always work, we hypothesized that network traffic alone (simple ping) is not enough for accurately choosing the game server with the best performance. Therefore, we investigated the possibility of CPU load of any game server affecting the playing experience for any players connected. Using our own server set up with Quake III, we came with a simple system of CPU loading we nicknamed the "Counter System." Using this system, versus just an ultra-loaded CPU, which would offer no form of load measuring, we were able to actually measure performance and ping while increasing load at a steady unit of measure. Simply loading a server would result in a reading of 100% (really 99%) CPU load at all times, thus creating no real relationship with ping and/or performance.

Using a simple C++ program:

```
#include <iostream.h>
…
int counter = 0;
while (1)
{
     ++counter;
}
…
…
```

We loaded the CPU of the server, while 5 bots and a human player competed. The map was always the same and the frag limit was set to 999 so that the game would not end by points in the middle of a test. As described in the following procedure, more

instances of counter.exe were used to increase the CPU load by “units” of load (a “unit” of load in this experiment is a “counter”).

4.2 Procedure

Though qstat was already set to be used, properly timing the instances using the command line would have not been efficient enough for our testing. Using the QStatExt tool, we were able to set qstat to run at a set interval (we chose every 10 seconds). 4 different experiments were used, as well as a control, to evaluate performance.

	Control	5 Counter	10 Counter	15 Counter	Increasing Lag
10	0	5	10	15	1
20	0	5	10	15	2
30	0	5	10	15	3
40	0	5	10	15	4
50	0	5	10	15	5
60	0	5	10	15	6
70	0	5	10	15	7
80	0	5	10	15	8
90	0	5	10	15	9
100	0	5	10	15	10
110	0	5	10	15	11
120	0	5	10	15	12
130	0	5	10	15	13
140	0	5	10	15	14
150	0	5	10	15	15
160	0	5	10	15	16
170	0	5	10	15	17
180	0	5	10	15	18
190	0	5	10	15	19
200	0	5	10	15	20
210	0	5	10	15	21
220	0	5	10	15	22
230	0	5	10	15	23
240	0	5	10	15	24
250	0	5	10	15	25
260	0	5	10	15	26
270	0	5	10	15	27
280	0	5	10	15	28
290	0	5	10	15	29
300	0	5	10	15	30

Table 2: Number of counter used per experiment

Table 2 shows how each of the experiments was run. Each experiment utilized 5 bots and one human player for five minutes (300 seconds). The first column in table 2 is the number of seconds elapsed after the human player joined the game. The next five columns show the number of counters running for each experiment at any given ten second interval.

After each of the experiments was run, the data collected (which included local ping, player ping to game server, number of players, and each player's score) was entered into Cumulative Distribution Function [CDF] graphs and compared local (server to server) ping versus player to server ping.

4.3 Analysis

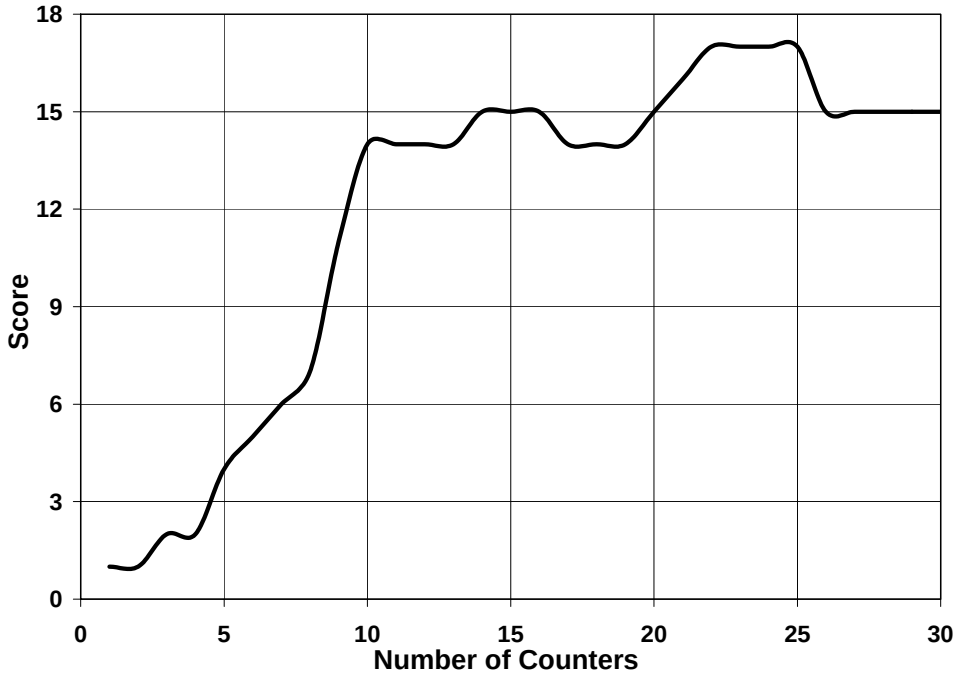

Figure 7: Player Score versus Active Counters in Increasing Load Experiment

The score was taken from the information retrieved by QStatExt, and is computed by adding 1 for every kill, and subtracting one for every self-destruct, or suicide. As shown in Figure 7, the human player's score stopped increasing significantly as the CPU load kept increasing. Using these graphs, we were able to make two very important conclusions. Firstly, CPU load does in fact affect gameplay, including kill rate and visual lag.

Figure 8 shows the distribution of local ping times reported by the server over time, with a specific number of counters running. Figure 9 shows the distribution of the player's ping times over time, also with a specific number of counter running. Using these CDF graphs for the control, 5 counter, 10 counter and 15 counter experiments, we were able to view the difference in the median ping times with higher CPU loads. By looking into the 90^{th} percentile, we found quite a few data points lying well outside of the average. Such data points could possibly be reported to the player if only one single ping is used, thus turning the player away from a server that is potentially good. However, it is also possible for a very bad server to be reported as good, since the median pings for most every server we tested were about the same, regardless of the number of outliers (high ping times) that existed. On the same token, the server's local pings were very similar despite the CPU load, whereas the players pings to the server were noticeably different the more loaded the CPU became. Consequently, these "tails" in the CDF graphs began to report larger sets of high pings, not accurately reflecting the performance of the server, with far more data points lying above the average as the load increases. Such an occurrence can result in badly reported ping times as well.

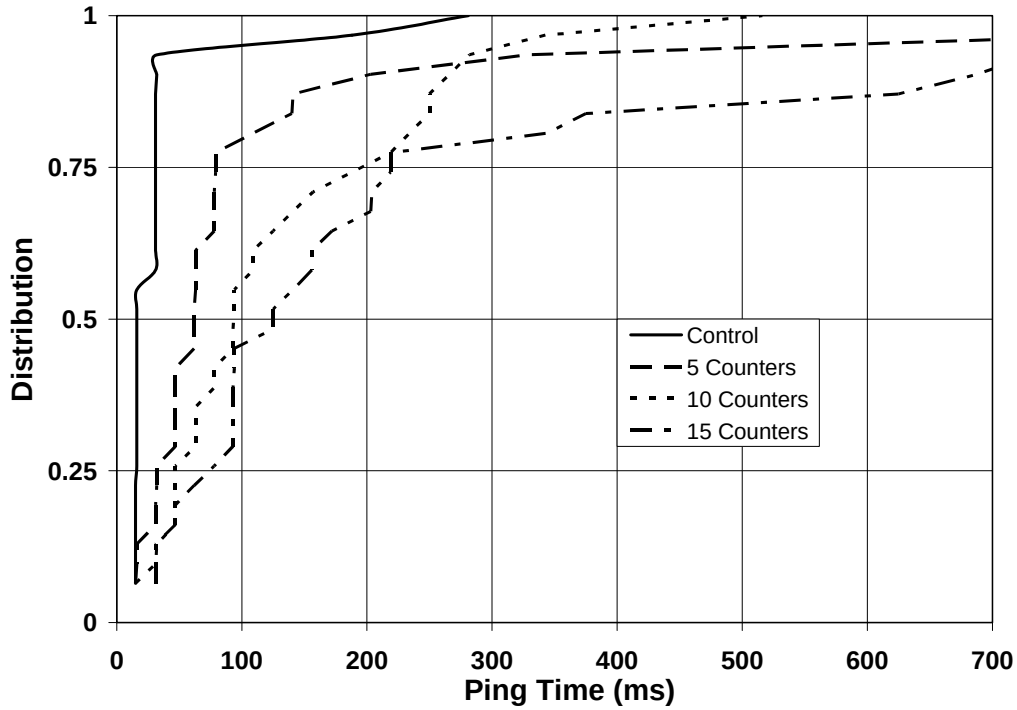

Figure 8: Distribution of Ping Times for Static Load Experiments (Local Ping)

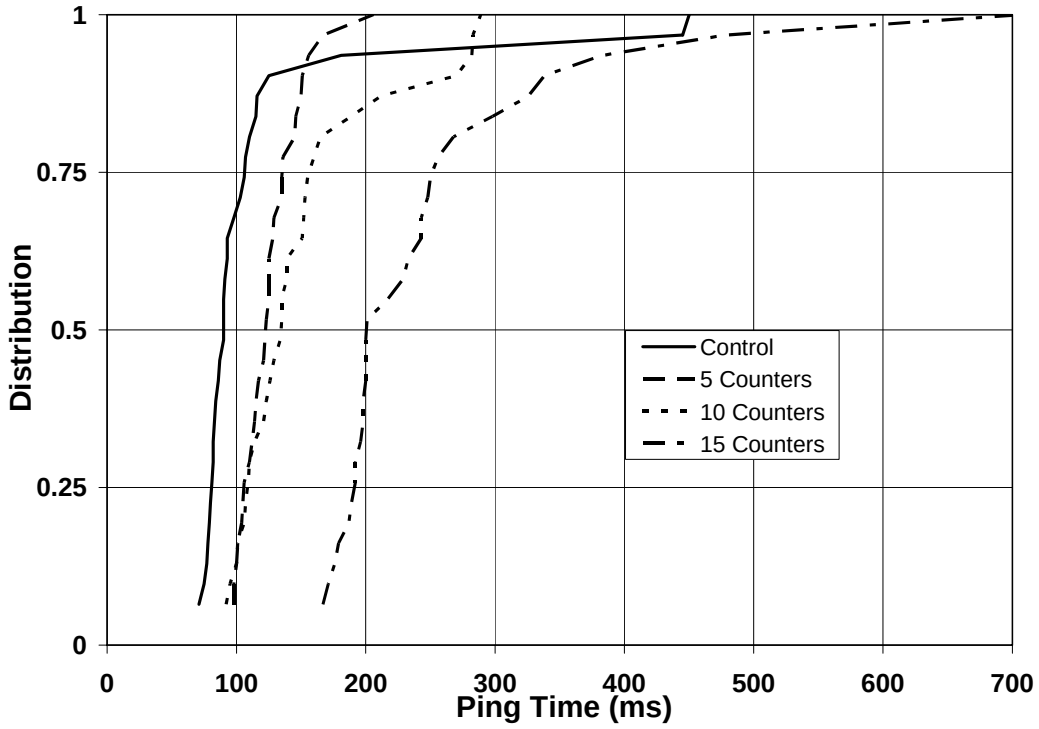

Figure 9: Distribution of Ping Times for Static Load Experiments (Player Ping)

Therefore, we concluded that a single ping time is not sufficient for the player to make a well-informed choice as to which server they should play on. It is actually a better idea to report a windowed average of ping times, in order to eliminate reporting network or server system anomalies in ping times. A windowed average of ping times, or a moving average, means taking the average of the reported ping time, and set of ping times before it, and reporting the average as the ping. The time span the average is taken in is known as the window. In other words, the player will never actually see ping times from the 90^{th} percentile, since they will be lumped into the moving average. The following table of values illustrates how this may work.

In Example 1, the Refresh List column represents when the server retrieved a local ping. Row 0 indicates when a player logged into the game, row -30 represents 30 seconds before the player joined and row 60 represents 60 seconds after the player joined. The table assumes the server is retrieving a local ping every 10 seconds. The Reported Ping column indicates the ping time that will be reported to the player when they list servers by ping time in milliseconds in the game's built-in server browser. Had the player pinged the server at time 40, they would have been presented with a ping of 478, and perhaps not have even seen the server in the list. However, if the ping time reported as a windowed average using 5 points:

Example 1:

Refresh List (s)	Ping Time Received (ms)	Reported Ping (ms)
-30	35	…
-20	30	…
-10	31	…
0 (start game)	33	…
10	30	31.8
20	34	31.6
30	28	31.2
40	478	120.6
50	32	120.4
60	33	121

Table 3: Server List Demonstration Using a Windowed Average

When a player joins the game server (at time 0 in this example), they see a moving average of 5 ping times, shown in the Reported Ping column. With the windowed average, the ping time of the server is not reported as a time that does not accurately reflect the performance of the server. After three more pings are received that are normal (closer to 30ms), the windowed average will decrease accordingly. In this way, it is possible for a "buffer zone" of pings to be reported, where for only a short while, the ping seems a bit higher, since such an occurrence may indicate a network traffic spike, while local ping will stay the same. The longer the spike lasts, the higher the average ping will rise, eventually coming back down to normal. It is a more accurate representation of the server's connection stability, as opposed to the local ping times that would not reflect such problems.

Example 2:

Refresh List (s)	Ping Time Received (ms)	Reported Ping (ms)
0	91	91
10	87	89
20	90	89.33
30	103	92.75
40	75	89.2
50	82	88
60	106	90.57
70	71	88.125
80	93	88.67
90	82	88
100	86	87.5
110	79	86.7
120	78	85.5
130	115	86.7
140	181	97.3
150	125	101.6
160	445	135.5
170	83	136.7
180	90	136.4
190	98	138
200	90	138.4
210	116	142.1
220	77	142
230	107	141.2
240	80	131.1
250	110	129.6
260	81	93.2
270	450	129.9
280	84	129.3
290	93	128.8
300	400	159.8

Table 4: Server List Demonstration Using a Windowed Average and Actual Data Points Collected via Experiment

Example 2 illustrates the same type of data as example 1, except using actual data points retrieved through our own experiments, and over a greater amount of time. With this data, we can see that the ping times as a whole move closer and closer to a steady trend (windowed average of 10 pings at a time).

5. MUSST

5.1 Motivation

There currently exists no good method for two players in two separate locations to choose a multiplayer game server which will perform well for both of them.

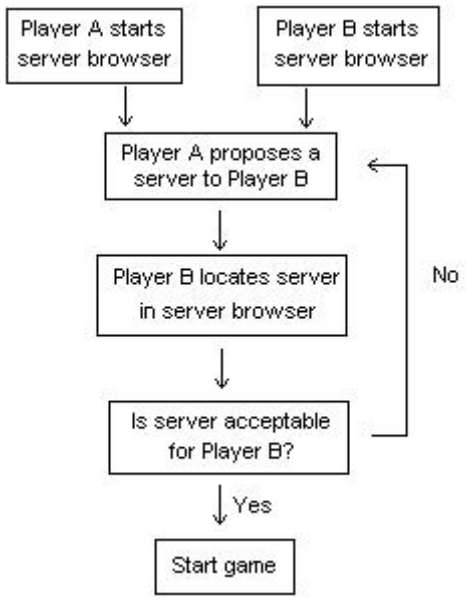

Figure 10: Manual Game Server Selection Flowchart for Two Players

At the present time, the only way for these two players to find a server which plays well from both locations is through an error-prone trial and error method, which also requires some sort of external communication mechanism. Assume Player A is connecting from San Francisco, California and Player B is connecting from Tokyo, Japan. Figure 10 depicts a flowchart of the steps each player would need to carry out before a (hopefully) appropriate server was found.

Player A uses the server browser of his game of choice to sort the list of servers by ping time. This value is the amount of time it takes a packet of data to travel from

each server's location to San Francisco, California. Player A will begin with the servers that have the lowest ping time and proceeding through the list to the servers with the highest ping times.

Player A will then suggest a server to Player B. The server with the lowest ping to San Francisco most likely does not have the lowest ping to Tokyo, therefore Player B will be required to scan through her list of servers until he is able to locate the specified server.

When both players have located the same server in their server browser, the next step is to look at the two ping values and decide if this server is acceptable. There are two possibilities at this point.

The first possibility is that the ping for Player B is also very low, in which case they have found an excellent server match. Player A and Player B can go ahead and connect to this server and should enjoy a high performance, low-latency game.

The second possibility is that the ping for Player B is unacceptable. Assume that the server in question is located in San Diego - a short trip to San Francisco, but clearly across the Pacific Ocean from Tokyo.

There are two basic options on how to proceed. Player A could move down on his server list and propose a new server to Player B, most likely the second best server on his list. At this point, the process discussed above would repeat itself with this new server. Player B would find the new server on his list, evaluate it, and make a decision.

The second option is for Player B to make a counter-offer, finding a server towards the top of his list and reporting it to Player A. Player A would locate the server

in his own server browser, check the ping value, and decide if the server is worth playing or not.

This process will repeat itself a number of times until a server is proposed by Player A and accepted as playable by Player B or vice-versa. This method is both time-consuming and inaccurate.

The number of servers available at any given time makes checking every possible server impossible, or at least impractical. Even evaluating a very a small percentage of these servers is time-consuming. Moreover, players will end up making a server choice which is almost definitely not the optimal choice due to the time it would require to evaluate all of the possible servers.

The situation discussed assumes that only two players are interested in playing together. As we add more users into this scenario, the problem becomes even more difficult one to solve.

5.2 MUSST Procedure

5.2.1 Pre-testing Phase

To test the accuracy of MUSST, four volunteers from different parts of the world were chosen to run the software locally, and connect to a running instance remotely located on a server at Worcester Polytechnic Institute in Worcester, MA. The volunteers were given the necessary software (including the *Multi-Location Server Selection Tool* and *qstat*) and installation instructions. The experiment was run at 14:00 Eastern Standard Time, and the volunteers were:

- Player A (Hoorn, The Netherlands) [82.217.15.247, Cable Internet Connection]
- Player B (MA, United States) [130.215.239.33, T3 Internet Connection]
- Player C (Kota Bharu, Malaysia) [60.48.62.140, ADSL Internet Connection]
- Player D (NC, United States) [67.77.3.251, SDSL Internet Connection]

Each of their IP's were recorded through an IRC DNS command, and confirmed by each player to insure correctness. The IP's were then WHOIS'ed (using the international and United States WHOIS databases) to ensure location accuracy.

5.2.2 Procedure

The tool was setup to listen for connections by the four players. Each volunteer was instructed to connect to IP 130.215.28.221 (the PC running the MUSST server, located in Worcester, MA) using the given software, each with a unique handle so each connection could be identified and linked to its respective player.

Upon receiving the four connections, the MUSST server instructed each client to run *qstat* and retrieve the correct information, which was then sent back to the hub. After processing the *Quake 3* server information of all four players, the Quake III server with the best ping time average for all players was chosen as "the best" for everyone, and that server IP was sent to each client.

5.2.3 Data Processing (Phase A – Visualizing Ping Times per Player)

MUSST produced XML files of all available servers and their pings for each player connected to it. Using this information, a scatter-plot was produced of Ping Time in milliseconds to each Quake III server versus the order the ping times were received by

each client (Figure 11). The ping order is the same for each client since it is provided by the Quake III master server (master3.idsoftware.com).

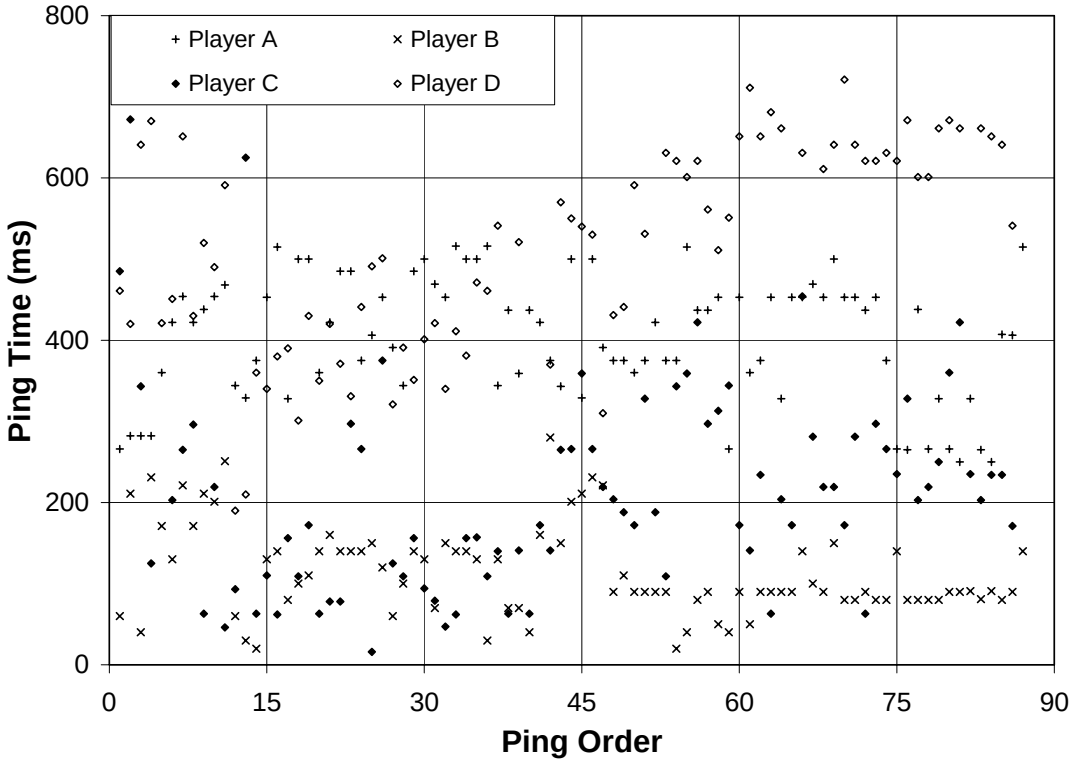

Figure 11: Scatter-plot of Ping Times of MUSST Testing Volunteers

The IP of the selected server was then highlighted in each player's server list, using the unique ping time the selected server returned to the player. The actual server selected is indicated with an arrow (Figure 12). In all cases, a relatively low ping was chosen for each player, despite the drastic differences in the game server ping time for each player.

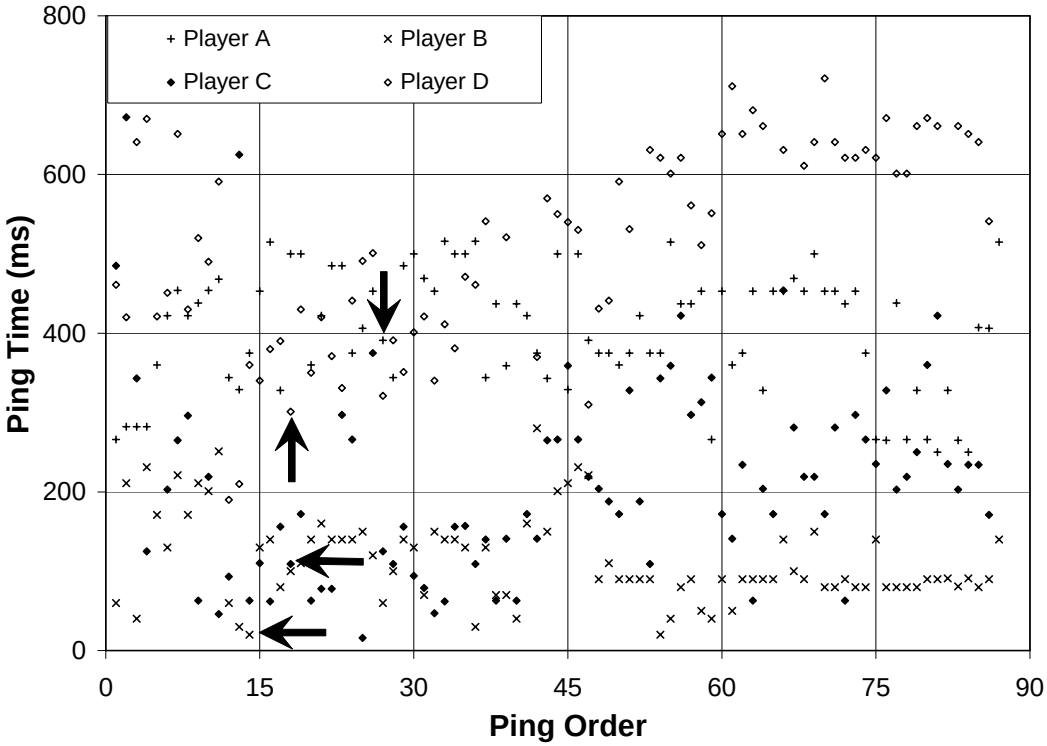

Figure 12: Scatter-plot of Ping Times of MUSST Testing Volunteers with Arrows to Indicate Server Chosen by MUSST

5.2.4 Data Processing (Phase B – Distribution of Ping Times)

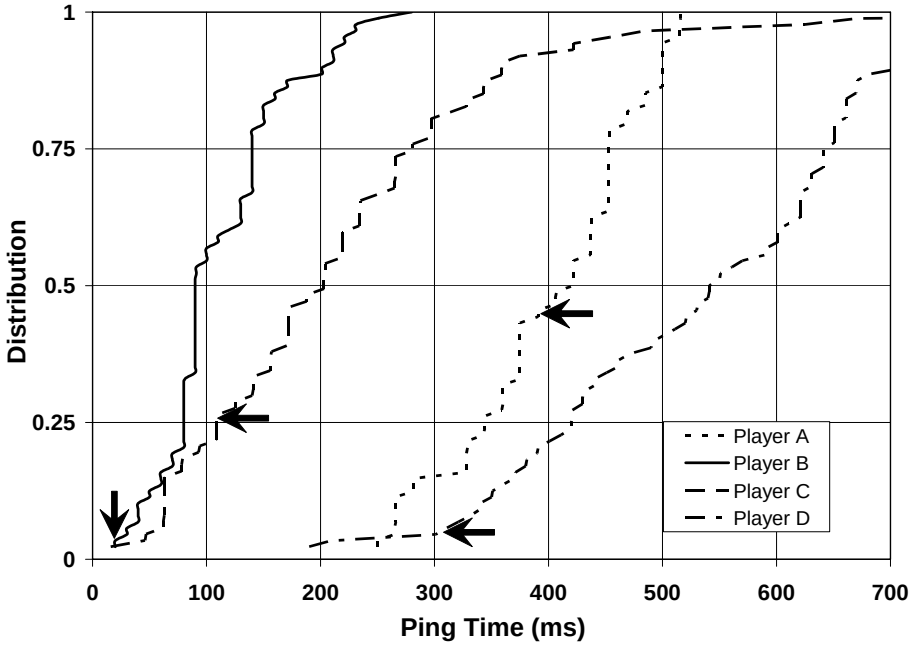

Figure 13: Cumulative Distribution Function of MUSST Volunteer's Ping Times

After tables were created with the data, and a scatter-plot was completed, a cumulative distribution function was then used to visualize the distributions of client ping times. On these graphs, the selected server is indicated with an arrow.

The representation show that no ping was chosen any higher than the 50th percentile, and no ping over 400ms was chosen either. Upon further review, it became apparent that the worst ping chosen (Player C) was picked to accommodate Player D and their high ping times to start with. So although the tool did not choose the absolute best server for each individual player, it still picked the "best" (in terms of lowest total times) server for all four players collectively, the original intent of the software.

5.2.5 Data Processing (Phase C – Visualizing the Physical Location of Clients/Server)

Through the use of domestic and international WHOIS databases, the locations of each player's DNS server and the chosen server's DNS server were pinpointed to a city. These locations were then plotted onto a world map, in order to get an idea of distances. The chosen server resided in New York City, NY, United States, as is plotted.

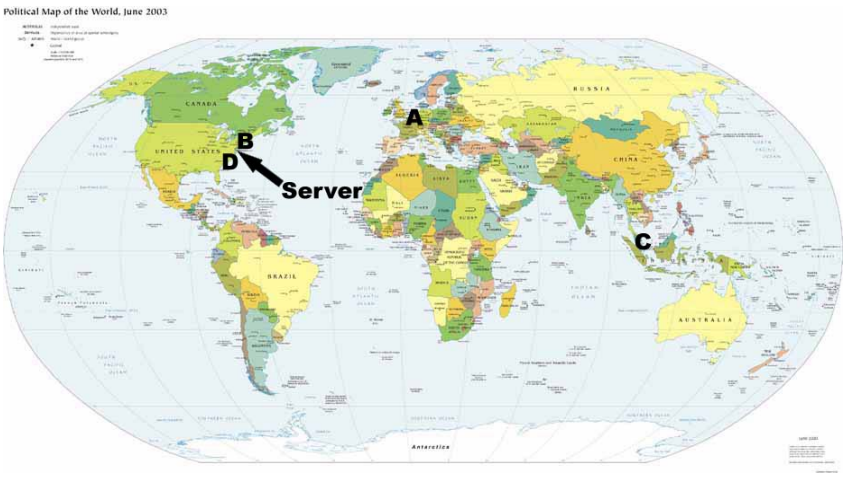

Figure 14: Map of the World Depicting the Locations of each MUSST Testing Volunteer and the Server Chosen by MUSST

6. Future Work

6.1 MUSST

The algorithm used to decide which server is best for a group of players given a complete set of ping data is an area which definitely requires further study.

This is an area with tremendous opportunity for optimization. Algorithm A could work best on games with lots of players whereas Algorithm B is best when looking for a smaller game. The algorithm may vary based on game size, time of day, and number of clients. Algorithm A may work better when looking for a Quake 3 server, whereas Algorithm B may work better when attempting to locate a Return to Castle Wolfenstein server.

There may also be differences based on geography. Perhaps certain assumptions can be made if the players happen to live in the same country, on the same continent, or within a certain distance of each other. For example, it may be possible to add a server list culling pass which tosses out some servers in the world based on their location, making the data processing steps faster and cleaner. It's also possible that certain data trends will emerge common to servers within a close proximity of each other, allowing the selection algorithm to be more accurate.

This tool could also be used as a completely different way to join a multiplayer game. Rather than functioning as a way for multiple known players to find a decent server to play on, MUSST could act as a virtual waiting room. The tool would sit on a known port of a known server and players from any location could connect to it at any time.

As soon as a certain player count threshold is reached, MUSST goes ahead and processes the players as it currently does, sending out qstat information and finally reporting the best server to play on. These players currently in the room would leave to play their game, and a whole new set of players would now connect to the MUSST server, and the process would repeat. This future possibility would require only a very small modification to MUSST, but to be of any practical use it would need a web-interface or some other way for players to quickly and easily location an appropriate MUSST server.

6.2 Local Load / Server Browser Extension

The local load server utility returns a few different statistic about the information gathered thus far in order to help the client make an intelligent decision about which server they should play on. At this point it is not known what piece of information is the most important to reflect to the user. Perhaps the ping time average is the most important, maybe it's the deviation of the pings, or it could also be the ninetieth percentile value. More work is required in this are to determine which statistic best corresponds to a playable server.

This further work also extends to the alternate server browser which keeps track of ping values locally and calculates statistics based on the data. This information is similar to the ping data described above, and similar studies will need to be done on this data to determine exactly how to decide which server is the best one to play on based on the known statistics.

6.3 Combination

The two sections of this project are separate solutions to separate problems, but they also are complimentary to each other. MUSST would also benefit from a complete server ping history; the tool could take these values into account when choosing a server for multiple players to use.

7. Conclusion

The increasing growth in number of online gamers recently has brought about an increasing need for additional analysis of how game servers, games and players coincide. Over the past years, many gamers have used the same method of server selection, which we feel has become an obsolete and inefficient way of selecting a gaming server. Moreover, the way game servers report their pings is also inefficient, and lacks the information necessary for proper server selection in the first place.

The more we continue to develop better games and more involved game worlds, the more we need to become conscious of the fact that while our technology is getting better, some of our methods and practices are not. For years, game server browsing has been an easy point and click technology, where game servers are sorted by ping time. Unfortunately, with more and more servers hitting the list, just the speed of their network at one specific point in time is not enough. We have seen that by taking a static ping time and using it to determine server stability, the player is left with a randomized selection, regardless of where the server may fall in the list, since ping time distributions show a set of outliers that would turn any player away, or even more importantly, attract the player to a server with poor performance.

Furthermore, as a server's CPU load increases, we find an even more interesting phenomenon. Although the number of outlying ping times may increase slightly, giving the player more of a chance of spotting the "bad server", the median ping times fall well within a safe range, much like a server with virtually no load.

So, it makes sense that a windowed average, as we have described, should become more commonplace for master game servers everywhere. Additionally, server

browsing is much more efficient using a browser akin to our Server Browser Extension tool, which slightly increases time to find a server in general, but drastically decreases time to find a “good” server.

While all of these ideas may be good for a single player, the growing industry should also offer options for friends and groups who always wish to play together, but never want to take the time to seek out a common server that performs well for all of them. Browsing servers manually and finding a good server for both players can take as long as checking the first server on the list, or searching every server on the list, and even longer with more than 2 players. Even with our Server Browser Extension tool, it is still difficult to accurately find a good server. Despite the tool displaying the windowed average, which we feel to be more accurate than a single ping time, there is still a large amount of time lost in waiting for the list to refresh multiple times, as well as manually figuring out the best server for multiple players.

Our MUSST tool is a definitive answer to this issue, and makes finding a common server simple. The use of such a program could and would help change the way player find servers, and possibly even increase playing time, after reducing server location time.

Therefore, one static ping is not enough for accurately choosing a good game server, and by continuing explorations into ping time, CPU load, etc., it seems that significant advances in server browsing among the online gaming community can be made.

8. References

[AMES] S. McCreary and K. Claffy. Trends in Wide Area IP Traffic Patterns: A View from Ames Internet Exchange. In *Proceedings of ITC Specialist Seminar on Measurement and Modeling of IP Traffic*, pages 1 – 11, Sept. 2000. [Online] http://www.caida.org/outreach/papers/2000/AIX0005/AIX0005.html.

[CSTRIKE] The Creators of Counterstrike in *Computer Gaming World*, Dec. 2003.

[ECODATA] Entertainment Software Association. Industry Sales and Economic Data. [Online] http://www.theesa.com/industrysales.html.

[ESA] Daniel Hewitt, Entertainment Software Association. Computer and Video Games Software Sales Break $7 Billion in 2003, Jan. 2004.

[IDSA] International Digital Software Association. Economic Impacts of the Demand for Playing Interactive Entertainment Software, 2001. [Online] http://www.isda.com/releases/EIS2001.pdf

[LATENCY] G. Armitage. An Experimental Estimation of Latency Sensitivity in Multiplayer Quake 3. In *Proceedings of the 11th IEEE International Conference on Networks (ICON)*, Sept. 2003

[LIVE] GameSpy. The Future of Live? [Online] http://xbox.gamespy.com/articles/559/559846p1.html

[MADDEN] J. Nichols and M. Claypool. The Effects of Latency on Online Madden NFL Football. In *Proceedings of the 14th ACM International Workshop on Network and Operating Systems Support for Digital Audio and Video(NOSSDAV)*, June 2004.

[MOD] David Kushner. *Masters of Doom*, 2003.

[NOTO] National Organization of Theatre Owners. Total US Box Office Grosses. [Online] http://www.natoonline.org/statisticsboxoffice.htm.

[PONG] The Pong Story. The Site of the First Video Game. [Online] http://www.pong-story.com/intro.htm

[PUNKBUSTER] PunkBuster Online Countermeasures http://www.punkbuster.com/

[QSTAT] S. Jankowski. QStat – Real-time Game Server Status. [Online] http://www.qstat.org

[QWORLD] QuakeWorld.net http://www.quakeworld.net/

[REALTIME] L. Pantel and L.C. Wolf. On the Impact of Delay on Real-Time Multiplayer Games. In *Proceedings of the Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV)*, May 2002.

[SALES] DesignTechnica. Halo 2 Breaks Retail Sales Record. [Online] http://news.designtechnica.com/article5906.html

[SALES2] CNN Money. Video Game Sales Jump 8 Percent in 2004. [Online] http://money.cnn.com/2005/01/18/technology/gamesales/

[SMB3] GameCubicle. Historical Units Sold Numbers for Mario Bros on NES, SNES, N64. [Online] http://www.gamecubicle.com/features-mario-units_sold_sales.htm

[TOP10] Entertainment Software Association. Top Ten Industry Facts. [Online] http://www.theesa.com/pressroom.html

[UT2K3] T. Beigbeder, R. Coughlan, C. Lusher, J. Plunkett, E. Agu, and M. Claypool. The Effects of Loss and Latency on User Performance in Unreal Tournament 2003. In *Proceedings of ACM Network and System Support for Games Workshop (NetGames)*, Sept. 2004.

[WCIII] N. Sheldon, E. Girard, S. Borg, M. Claypool, and E. Agu. The Effect of Latency on User Performance in Warcraft III. In *Proceedings of ACM Network and Systems Support for Games Workshop (NetGames)*, May 2003.

[WOW] Blizzard Entertainment. World of Warcraft Shatters Day-One Sales Records. [Online] http://www.blizzard.com/press/041201.shtml