UNIVERSITY OF OSLO
Department of Informatics

# Monitoring and Analyzing a Game Server Scenario

Stian Opsahl Jelmert

Network and System Administration

Oslo University College

May 19, 2008

# Monitoring and Analyzing a Game Server Scenario

Stian Opsahl Jelmert

Network and System Administration
Oslo University College

May 19, 2008

**Abstract**

Today, most literature about services in system administration is about conventional services like email servers. How could one monitor and analyze a scenario where the service in question is a game server? As these two services are technologically different, conventional monitoring tools may miss vital information in the context of game servers.

This thesis focuses on developing a monitoring system for a game server in order to learn and understand the characteristics of a game server process in a production environment. An experiment is carried out to control some of the influencing variables, like the number of players and game server instances, and to observe the system under the different conditions. Results show that the number of instances did not affect the overall performance in that way we expected. The concurrent players on the server dominates the CPU load. We find that a strong linear relationship exist between these two variables. When it comes to memory usage, players barely affect this resource in our experiment, but the number of game server instances (Team Fortress 2 dedicated server) does. The server process allocates most of its needed memory in the beginning. The amount allocated depends on which map is set when the game server is executed.

This study has shown that the game server, in this case Team Fortress 2, is a predictable service in terms of resources.

# Acknowledgements

First and foremost, I would like to thank my supervisor, assistant professor Hårek Haugerud for his support and guidance throughout this semester. Your help has been much appreciated. A special thanks goes to Kyrre Begnum for his enthusiasm, fruitful discussions and coming up with a project idea related to my main interest, games!

I would also like to thank the students at Oslo University College whom volunteered and Valve Corporation, especially Arsenio Navarro at Academic Licensing for making the experiment possible.

Finally, i would like to thank my family for motivating talks and for believing in me. Least but not last, Hanne for bearing out with me in this stressful period. Also for her patience and endless support.

Oslo, May 2008.

Stian Opsahl Jelmert

# Contents

**Acknowledgements** **ii**

**1 Introduction** **2**
1.1 The Gaming Industry . . . 2
1.2 Game Server Provisioning and Performance . . . 3
1.3 Motivation . . . 4
1.4 Problem To Be Addressed . . . 4
1.5 Approach . . . 5
1.6 Thesis Outline . . . 5

**2 Background** **6**
2.1 Conventional Monitoring Tools . . . 6
2.1.1 Munin . . . 6
2.1.2 Nagios . . . 6
2.1.3 Cacti . . . 7
2.2 Game Server Tool . . . 7
2.3 Performance Monitoring and Analysis . . . 7
2.4 Games Are Important . . . 9
2.4.1 The Industry . . . 10
2.5 The Game . . . 13
2.5.1 Getting Started . . . 13
2.5.2 The Gameplay . . . 14
2.5.3 The Architecture . . . 15
2.6 Related Work . . . 17

**3 Approach** **20**
3.1 The Scientific Method . . . 20
3.2 Set Up A Real Life Service . . . 22
3.2.1 Make the Server Attractive . . . 22
3.3 A Monitoring Framework . . . 24
3.3.1 Online Data Collection . . . 25
3.3.2 Offline Data Analysis . . . 31
3.4 Controlled Experiment . . . 33
3.4.1 Hardware Equipment and Software . . . 33
3.4.2 Preparations In Advance . . . 33
3.4.3 Performing The Experiment . . . 35

**4 Results** **38**

4.1 Results From The Questionnaire . . . 38
4.1.1 Scenario 1 . . . 38
4.1.2 Scenario 2 . . . 38
4.1.3 Scenario 3 . . . 40
4.2 Results From Experiment . . . 41
4.2.1 Players . . . 41
4.2.2 CPU% . . . 42
4.2.3 Memory% . . . 48
4.2.4 Resident Set Size . . . 50
4.2.5 Minor Faults . . . 51
4.2.6 Virtual Memory . . . 52

**5 Discussion** **54**
5.1 Server Capacity Planning . . . 54
5.2 Impact . . . 55
5.3 Review of The Questions . . . 56
5.3.1 How many game servers can run simultaneously on one machine? . . . 56
5.3.2 What is the bottleneck that stops us from running one more server? . . . 56
5.3.3 How predictable is a game session in form of resource use? . . . 56
5.3.4 What characterize a server which has no resources left? . . . 57
5.3.5 The best time of the day doing maintenance? . . . 57
5.4 Reliability and Validity . . . 57
5.5 Repeatability . . . 58

**6 Conclusion and Future Work** **60**
6.1 Future Work . . . 61
6.1.1 Munin . . . 61
6.1.2 Other Games and Hardware . . . 61
6.1.3 Improve Monitoring System . . . 61

**A Setting Up a TF2 Server (linux)** **I**

**B Game Configuration Files** **III**
B.1 Server Configuration (server.cfg) . . . III
B.2 Message of The Day (motd.txt) . . . VI
B.2.1 motd.hml . . . VII
B.3 Map Cycle (mapcycle.txt) . . . IX
B.4 Autoexec (autoexec.cfg) . . . IX

**C Miscellaneous Installs** **XI**
C.1 Install New Kernel . . . XI
C.2 Set Up Web Server . . . XI
C.3 Enable public_html . . . XI

**D Emails** **XIII**
D.1 Academic licensing at Valve . . . XIII

D.2 Invitation To Game Evening . . . . . XV
D.3 Response To The Requests . . . . . XVI

**E Letters** **XIX**
E.1 Experiment at School . . . . . XIX
E.2 Questionnaire . . . . . XXI

**F Scripts** **XXV**
F.1 Shell Script . . . . . XXV
F.1.1 execute.sh . . . . . XXV
F.2 Perl Script . . . . . XXVIII
F.2.1 datacollection.pl . . . . . XXVIII
F.2.2 analyze.pl . . . . . XXXI
F.2.3 update.pl . . . . . XXXIII

# List of Tables


3.1 Time variable . . . . . 25
3.2 Process Status variables . . . . . 25
3.3 Proc variables . . . . . 27
3.4 Game variables . . . . . 28
3.5 Specifications of the server . . . . . 33
3.6 Specifications of the clients . . . . . 33
3.7 An brief overview of the execution process. . . . . 36

4.1 Server instance in each scenario. . . . . 41
4.2 Correlation coefficient and coefficient of determination for each server instance. 46
4.3 Results from the twenty tests. . . . . 49

# List of Figures


2.1 Illustrates a white box approach . . . . . 7
2.2 Illustrates a black box approach . . . . . 8
2.3 The server browser in Steam. . . . . 14
2.4 Featuring the character Demoman at RED team on the map named cp_dustbowl. 15
2.5 Illustrates a client-server architecture . . . . . 16

3.1 Illustrates the process of collecting and analyzing data . . . . . 24
3.2 Establishing a socket connection to game server . . . . . 29
3.3 Unpack response packet. . . . . 29
3.4 Shows player activity on Tuesday. . . . . 29
3.5 Shows player activity on Wednesday. . . . . 30
3.6 Player activity on the game server, week 10th. The figure shows a a similar activity pattern for Monday, Tuesday and Wednesday. . . . . 30
3.7 If the response packet is not received after four seconds, fixed values are set. 31
3.8 Example of an auto generated web page for the proc23 log file . . . . . 32
3.9 Illustrates the classroom where the experiment takes place. . . . . 35

4.1 How smooth did the game run while playing? . . . . . 39
4.2 If you were playing on this server in leisure time and the playing was not a part of a experiment, would you still continue playing or change to another server? . . . . . 39
4.3 How smooth did the game run while playing? . . . . . 39
4.4 If you were playing on this server in leisure time and the playing was not a part of a experiment, would you still continue playing or change to another server? . . . . . 40
4.5 How smooth did the game run while playing? . . . . . 40
4.6 If you were playing on this server in leisure time and the playing was not a part of a experiment, would you still continue playing or change to another server? . . . . . 41
4.7 Comparing the player count in each scenario. . . . . 42
4.8 Comparing the CPU% in each scenario. . . . . 43
4.9 Total CPU usage by server1. . . . . 44
4.10 CPU usage by server2, server3 and total usage. . . . . 44
4.11 CPU usage by server4, server5, server6 and total usage. . . . . 45
4.12 Scatter diagram of the real life data sample. . . . . 47
4.13 Estimation of a new measurement value with a 95% prediction interval (PI). 48
4.14 Comparing the MEM% in each scenario. . . . . 49
4.15 Comparing all official maps impact on MEM%. . . . . 50

4.16 Comparing the RSS in each scenario. . . . . 51
4.17 Comparing the minor fault in each scenario. . . . . 51
4.18 Comparing the VSZ in each scenario. . . . . 52

B.1 What a user will see after connecting to OUC's Team Fortress 2 server . . . VI

# Chapter 1

# Introduction

*"The video game industry is entering a new era, an era where technology and creativity will fuse to produce some of the most stunning entertainment of the 21st Century. Decades from now, cultural historians will look back at this time and say it is when the definition of entertainment changed forever."*

*Douglas Lowenstein[1]*

## 1.1 The Gaming Industry

Change has always been a keyword in the game industry. Game developers seek to continuously innovate and produce new games with great game play and stunning graphics as they compete in a highly competitive marked. The computer- and video game industry, also referred to entertainment software industry is growing rapidly. In 2006, 204.7 million units of computer and video games were sold. The gaming industry in USA alone took in 7.4 billion dollars in 2006, according to The Entertainment Software Association (ESA)[2].

The industry is not only important for those who are involved in making games, but also for those who subsist on it, like complementary markets. A complement is a product which one might buy as an addition to an already established product. For example consumers buy an electronic device like a racing wheel with gas and brake pedals[3] to enhance the gaming experience in racing games. Crandall and Sidak[4] divide complementary products of entertainment software into four groups: processor, content, electronic devices and bandwidth.

Game Server Provider (GSP) is a result of the entertainment software industry. They earn money by leasing out game servers. Typically they offer two types of game servers, private and public. Public servers are commonly owned by game communities, and is not password protected. This implies that they are available for everyone who wish to enjoy a online game with others. A private game server is a password protected server, and usually owned by clans. A clan can be described as a group of players who play together on a regular basis in a specific game, usually motivated

by common interest or goal. The clans uses their game server to play PCW (Practice Clan War) matches against other clans. Some play for fun while others play on a professional level in ranked leagues[5], such players are referred to as cyber athletes.

Multiplayer online games are different from traditional single user games since the game is shared with other players. Quake, released in 1996 by id Software, was the first game to support multiplayer over the internet. The game belongs in the First Person Shooter (FPS) genre. In a FPS game, the player takes a first person perspective, hence the name. Multiplayer games belonging to this genre are often ”session based”, which mean that the users play a map for a certain amount of time and then move on to a new map. Another characteristic is that they are depended of the low Round Trip Time (RTT). RTT is the time a packet with information describing the player's action (e.g. keyboard input) spends on to get to the server, and then back again to the client with information describing the result of the player's action. If the packet gets delayed, the player will experience so called ”lag” (the game stutters), which again affects the player experience of the user. Therefore players seek out game servers offering the lowest RTT. A study by Chambers et al.[6] shows that gamers as customers are extremely difficult to satisfy since they lose interest if their expectations are not met. For GSPs it is crucial in order to reach business goals to provide services which satisfy their customers' expectations.

## 1.2 Game Server Provisioning and Performance

Abdelkhalek et al.[7] states two conditions for successfully providing both traditional and novel services (e.g. game server) to a great number of clients. The first is enhanced networks, while the latter is powerful servers. A game server requires fast CPU, RAM and high bandwidth for optimal game play. The server must be capable of processing the data from all connected clients on the server and then send it back to them. Understanding how a running service scales is crucial in order to support it[7], and monitoring is a way of of gaining this insight.

Monitoring the performance of a running service is one of many important tasks for a system administrator. By monitoring the service you receive ”up to date information” about the behavior of system resources. This makes it easier to detect when problems occur and to determine the cause[8]. Picture yourself in a situation where you are running a game server hosting company, what information is important for you to know in order to run a successfully business? What metrics are considered important? You may want to know the best time of the day doing maintenance. If there is a need for provisioning, when is the server stable? When will the service become degraded? Is there a possibility to run other services or multiple game servers simultaneously to increase revenue?

Are monitoring tools used today capable to assist answering these questions? Today, most literature about services in system administration is about conventional services like web- and email servers. How could one monitor and analyze a scenario where the service in question is a game server? A dedicated game server and an email server serve the same purpose, to serve their clients. However, these two services are

technologically different, thus place emphasis on different factors. Therefore existing tools like Munin, Nagios and Cacti may miss vital information in the context of game servers. Today, most of the available tools for game servers seek to provide game play statistics (HLstatsX[9]) and ease of in-game administration (Half-Life Server Watch (HLSW)[10]). HLstatsX, along with serverspy.net and game-monitor.com does however provide some graphs, but these are limited to player count on the given server. It is reasonable to believe that most companies involved in game server hosting have their own customized scripts for monitoring the performance of their own servers.

## 1.3 Motivation

In the context of computers, games are truly an area of importance. The author considers himself as an active gamer, playing computer games since the discovery of dad's 286 in the late 80's. He has experienced the evolution of gaming technology on both PC and on third to seventh generation consoles as graphic changed from simple one dimensional pixel, to vector, polygons and till now High Definition. During the period of study the author has realized that today, most literature about services in the context of system administration focus on conventional services like web- and email servers, but not game servers as a service.

## 1.4 Problem To Be Addressed

Based on the motivation above the following statement is formulated:

*How can one make a monitoring system for a game server?*

Some of the terms can be described further:

- A Monitoring System, means in this context a ”performance monitoring and profiling” system of the game server. The system should both collect and analyze important data at regular intervals.
- Game Server: A computer which is set to run one or more server applications. In this case, the game application is a Source Dedicated Server (SRDS) instance. It is common practice to name the computer after its running application, therefore this computer will be referred to as ”game server”.
- How can one make, focus on identifying the right variables and which methods of analysis that can give better information or so called decision support.

## 1.5 Approach

The approach of solving a particular problem often reflects a person's way of thinking. The author's approach is divided into four phases. In the first phase a real life service is installed and configured. The next step deals with creating a monitoring framework. The framework should be easy to use, flexible and contain many process related variables. In phase three the author will analyze real life data captured from the game server through the monitoring tool and try to identify important variables. In the very last phase a controlled experiment will be conducted to test variable usefulness by addressing the following five questions:

1. How many game servers can run simultaneously on one machine?
2. How predictable is a game session in form of resource use?
3. What characterize a server which has no resources left?
4. What is the bottleneck(s) that stops us from running one more server?
5. The best time of the day doing maintenance?

These questions are considered to be important for successful maintenance. Although there may be other questions which are equally important, the author choose to focus on the mentioned.

## 1.6 Thesis Outline

**Chapter 2: Background** This chapter gives a brief presentation of some monitoring tools for both conventional and novel services. The author also presents performance goals and techniques, importance of the entertainment software industry, the game and related work.

**Chapter 3: Approach** This chapter provides an in-depth description of the approach taken in order to answer the problem statement.

**Chapter 4: Results** This chapter presents and compare the obtained results from the experiment. Additionally, in some cases the results are compared to real life data and new measurements are carried out.

**Chapter 5: Discussion** In this chapter we discuss the important findings of the result chapter and its impact. We review the five questions raised in the introduction. We examine the reliability and validity of the measuring, and look at the repeatability of the experiment.

**Chapter 6: Conclusion and Future Work** In this chapter we answer the problem statement and make suggestions for future work.

**Appendix** This chapter contains all information related to the project. It includes emails, letters, scripts, game configuration files and various installs.

# Chapter 2

# Background

This chapter gives a brief presentation of some tools used in monitoring services. The author also presents goals and techniques in the context of performance analysis, the importance of the entertainment software industry, the game used in this study and related work.

## 2.1 Conventional Monitoring Tools

### 2.1.1 Munin

Munin[11] is a monitoring tool based on the RDDtool. The program collects data from one or more hosts. The program runs a master/node architecture. The master host runs munin and connects to all nodes (host running munin-node) and then pulls out data. Data are pulled out every five minutes through a cron job:

```
*/5 * * * * munin if [ -x /usr/bin/munin-cron ]; then /usr/bin/munin-cron; fi
```

and stored as RDD files. Based on the information it generates plots (graphs). These graphs are made available through a user friendly web interface, usually http://localhost/munin. Also, the graphs can be viewed by day, week, month and year. By default it provides information about disk, network, processes and system, but premade plugins supporting other areas are easily added as well as new custom plugins.

### 2.1.2 Nagios

A tool for monitoring the hosts in a network and its running services. Nagios[12] provides many features, i.a. monitoring host resources, running processes, alert notification (through e.g. email and pager) and trend analysis. Like Munin, all information is available through a website.

### 2.1.3 Cacti

Cacti[13] shares many similarities with Munin as it is based on the RDDtool and data is pulled out at interval of five minutes through a cron-job. The graphs are displayed on a website. The tool can pull data from small setups with few hosts up to thousands. As opposed to Munin it features various ways to view (list view, preview mode and tree view) the graphs and user management, allowing a added user to e.g. change parameters on graphs through rights.

## 2.2 Game Server Tool

Half-Life Server Watch is developed by Timo Stripf[10]. The tool provides none performance monitoring functionality, only a overview of server settings, ping and players currently on the server. It is the only tool that provides remote administration of a game server. This is made possible through Steam's Remote Control (RCON) and server query protocol[14]. RCON provides a user with administrator rights on a server, the right to exercise authority. E.g. kick players that do not follow the server rules, change map, change configuration etc. This feature is enabled in the program by typing the predefined RCON password of the server.

## 2.3 Performance Monitoring and Analysis

The performance of a system depends on how well resources like CPU, disk, memory are applied to the existing demand for them by jobs in the system[15]. There are three different way to test a system. These are referred to as black, white and gray box testing. In white box, (aka clear box, glass box and structural[16]) the internals are important (figure 2.1). Developing this kind of testing requires knowledge about the functioning(e.g. the structure and logic of a code[17])of the system.


Figure 2.1: Illustrates a white box approach

Examples from white box testing is unit- and security testing. While the black box approach can be done by a regular tester, the white box requires a skilled tester with in-depth knowledge. Grey box (aka translucent-box[16]) testing is a combination of the black and white box technique and could be used in cases where white- or black box alone is considered insufficient.

Black box (also referred to as opaque tests, behavioral, functional and closed-box[16]) treats the system as a black box. This means that we are only concerned about what

comes in (input) and what goes out (output), and not what happens in between (internally) (figure 2.2).


Figure 2.2: Illustrates a black box approach

Consequently we do not know how the system derives its output[17]. The testing is successful if the external input gives the expected results. Typical black box scenarios is load-, stress- and recovery testing. Monitoring a game server's performance or network traffic usually takes a black box approach. Because it is not always so that the person whom designs the tests has the luxury to access source code or the required knowledge to implement white box testing. Related work[18] has shown that developing monitoring scripts to collect data as input is a way to do it.

The reason for analyzing the performance of a computer system are motivated by different goals. Lilja[19] mentions six typical goals:

1. Evaluate different alternatives: For instance comparing several desktop computers. Each of these computers are specifically made for ”gaming” and their manufacturer promise top notch gaming experience. However, the are shipped with different hardware which may affect the performance. By carrying out a analysis, the analyst can provide information (in terms of numbers) of which computer that performs best under various conditions.

2. Evaluate a feature's impact: E.g. find out the impact of upgrading the GPU to a better one. To do so the analyst must carry out analysis before and after changing the component.

3. Tuning the system: Find the parameter values which gives the best overall performance.

4. Relative performance: Quantify how the performance of a system has changed in respect to older computer system.

5. Application performance issues: The process of making the program e.g. XYZ is now ”done”. The program does what is is designed to do, however the performance of the program is poor. In this case the analyst must apply relevant tools and techniques to locate the cause so it hopefully can be corrected.

6. Expectations: Set realistic expectations for what a next generation computer system or e.g. video game console is capable of doing.

Simulation, measurements and analytical modeling are three techniques that can be used to solve a performance analysis problem. A simulation is a imitation of something

real. In the field of computers systems, simulator is a program which is designed to simulate important components of a system. The approach is is highly flexible as the simulator can be modified with ease to study what happens when each of the components are changed. Also, cost are reduced as developing a simulation program is likely to be cheaper than purchasing the actual machine, even though developing simulation programs are time-consuming.

Measurements are as opposed to simulation, working on a physical machine. The technique is not as flexible as simulation, but provides real results[19].

The last technique describes the system by using mathematics. According to Lilja[19], it is less accurate and believable, but is useful as it provides insight. This insight can be used to carry out more detailed experiments with the two other techniques[19]. Clearly, each of these techniques has its own strengths and weaknesses.

## 2.4 Games Are Important

It all started with ”tennis for two” in 1958, a simple video game which simulated a tennis match. It was developed on an oscilloscope by William
Higinbotham[20]. The computer technology which made this possible was originally used to create missile simulations during the cold war. According to Higinbotham the intention behind the game was to entertain visitors visiting Brookhaven National Laboratory[21].

Four years later Steve Russell et al.[22] made ”Spacewar!” on a PDP-1 [1] at the MIT. However, this game was an open source game, thus not sold. Nevertheless it played an important part for two reasons: the invention of joystick and the first game to put destruction on the screen[23]. A similar game called ”Space Travel”, developed by Ken Thompson in 1969 should also be mentioned in this context. The game took place in outer space where the player controls a spaceship flying around in the solar system. The game was originally written on MULTICS. Thompson was a part of the AT & T Bell Labs staff which worked on new a multi-user OS called Multiplexed Information and Computing System (MULTICS) along with General Electric and MIT. After being pulled out of the project, the game was ported to FORTRAN on the GECOS OS. Playing ”Space Travel” on GECOS resulted in poor game experience. Additionally, it was expensive, 75$ per hour[24]. Therefore, Thompson and Ritchie rewrote the program to run on a PDP-7 computer. The machine was superior in terms of display processor compared to the former[25]. Not long after, Thompson et al. implemented a file system on the PDP-7, then came the user-level utilities (e.g. copy, delete and edit files) and least but not last the shell[25]. As the new OS took shape it was clear that it supported only one user. This led Brian Kernighan to call the OS for Uniplexed Information and Computing System (UNICS) as a joke to MULTICS[24]. UNICS was later changed to ”UNIX”.

Ralph Baer made the first generation video game console system[26], called Magnavox Odyssey in 1972. But the birth of the game industry came with Atari and their game Pong in 1972[27]. At that time the very same company behind a console also

[1] Programmed Data Processor-1, produced by Digital Equipment Corporation in 1960.

developed games for their console, but this changed with Nintendo's game license fee model. This implied that third party developers had to pay a license fee to Nintendo where they in return both tested and produced their games. They also controlled (among other things) how much a developer could earn, even though the demand for the game was high and the sales were going well[28]. The next step in the evolution of game industry was game developers receiving royalties.

### 2.4.1 The Industry

The computer and video gaming industry is often referred to as the entertainment software industry. The term "entertainment software" includes PC, console, online and wireless games[4] and is described in a paper by Hickling Arthurs Low[29]as:

> *"Entertainment software refers to interactive, software-based games that are played on a variety of electronic platforms with display devices (typically screens), sound reproduction capabilities, input interfaces such as keyboards, joysticks, and mice. These games combine narrative, sophisticated visual representations, music and sound, artificial intelligence, and often interaction with other players to produce unique entertainment experiences."*

R. Crandall and J. Sidak[4] mention three prominent economic characteristics about this industry:

1. Cyclic nature: The cycle lasts between five to seven years. The demand for software hits the highest point one to two years after the peak demand for the related hardware. Thereafter the demand slowly decreases as one waits for the next generation.

2. First-mover advantage: the company who first releases a new generation console to the marked will likely benefit from this by establishing marked share before rival companies can release their console. An example is the release of seventh generation console Xbox 360. Microsoft released their console a year before Sony and Nintendo did, thus had a lead in marked share, before Wii finally caught up[30].

3. Network effects: As number of users' increases to the given console, thus more software titles will be produced for that console.

In 1996 there were sold 74.1 million units of computer and video games in the America. Each year the unit sales increased considerably. In 2006, 204.7 million units of computer and video games were sold. To illustrate the importance of these numbers, the gaming industry in USA took in 7.4 billion dollars in 2006 based on the sales, according to The Entertainment Software Association (ESA) [2]. Except for the year 2005 (made 7.0 billion), historical sales charts from the NPD group show us a continuously increasing graph of units sold and dollars made in the past ten years[1]. Note that

these numbers considers domestic sale only, not taking profit from exporting games to foreign countries in their estimate. A similar study reports that entertainment software from U.S firms exported to other countries for 2.1 billion and domestic sales reached 8.2 billion dollars in 2004[4], which obviously does not correspond to ESA's 7.4 billion dollars for the same year [2]. Nevertheless, the sales are expected to increase the upcoming years. Michael Gallagher, a Chief Executive Officer (CEO) of ESA speaks about the importance of the game industry to the economy[31]:

> *"Computer and video game companies play an ever increasing role in our nation's growing economy. These companies and their colleagues across the nation are making entertainment software one of the fastest growing industries in the United States."*

According to Siwek[32], a principal at Economists Incorporated and the author behind "Video Games in the 21st Century: Economic Contributions of the U.S. Entertainment Software Industry", the game industry's value added to U.S. Gross Domestic Product (GDP) rose from 2.6 billion dollars in 2002 to 3.8 in 2006. The real growth rate in 2003-04 and 2005-06 was greater than 17.0% compared to US economy which was less than 4.0%[31]. GDP is a way of measuring the size of a country's economy and can exemplified by that for every game bought in the U.S. contributes to the country's GDP. GDP is defined by wikipedia[33] as:

> *"(...) the total market value of all final goods and services produced within a country in a given period of time."*

The numbers presented above do not tell us the true economic footprint of game industry. The structure of entertainment software consists of several parts[4]. These are called input, production, complements and output. Input is necessary for the production of a game, the production gives growth to complementary products and output is technological transfer to other industries. Each part contributes to the economy.

### Input

The process of creating a game requires input such as labor, research and development (R&D), advanced computers and capital. Typical jobs are animators, artists, programmers, level designers and marketing personnel. Siwek[32] reveals that the US entertainment software industry directly employs more than 24.000 people (in 2006) in 31 states and total 80.000 people when taking both directly and indirectly employment into account. The employees' uses powerful workstations optimized[34] for game design and customized input devices in their making, not "off the shelf" products targeted for the general masses because e.g. creating characters and rendering graphics consumes a lot CPU power. To make good games even better, a lot of effort is put in the R&D field to make groundbreaking AI (Artificial Intelligence) or graphics (game engine) which is very important these days to seize the market.

### Complements

Demand for an entertainment software product will lead to increased demand for complementary products or speed up the introduction of such products if non existing . For instance it is likely to assume that the release of HD consoles (e.g. PS3) increased the demand after High-Definition Television and Blu-ray movies. R. Crandall and J. Sidak divide complements into four parts; processors, devices, content and bandwidth because entertainment software affects each of them.

Each year games become more advanced and the requirements to run the game likewise. One can say that the game industry drives the development of faster processors (Central Processing Unit (CPU) and Graphical Processing Unit (GPU)) as they produce games that always push or goes beyond the performance of existing technology. The demand for more complex games increases the demand for companies like Intel, AMD and IBM to develop, produce and continuously improve their technology.

Entertainment software generates demands for other types of electronic devices. When you buy a game you need something to make it run, this could be a PC (specialized, off-the-shelf or homebuilt)or a console (stationary or handheld). According to VG Chartz, Wii has sold 19.94 million units, while Xbox 360 16.64 and PS3 9.40[30]. You also want something to display the game, like an HDTV or a LCD screen. A Nielsen Company study presented at CEPro[35] shows that HD display is connected to 71% of the PS3, while Xbox 360 had 66%. For enhanced in-game experience you might want to drop the poor sound quality which a television gives and buy surround system in addition to interactive devices such as microphone or special purpose controllers (steering wheel and foot pedals and guns). The same study[35] also reveals that 54% of PS3 and 48% of Xbox 360 owners are connected to 5.1 surround system or greater (7.1).

Online games consume bandwidth, but so does the demand for downloadable content (videos, game demos etc.) from the net. ISP companies providing bandwidth benefit from entertainment software as many released games do require an internet connection. Two of the most popular online games till now are the Massive Multiplayer Online Role-Playing Game (MMORPG) World of Warcraft and the First-Person Shooter game (FPS) Counter Strike. In 22 January 2008 WoW reached 10 million subscribers since their release[36]. Taking into account that each user pays a monthly, half or one year fee, there is a lot of money in circulation. Games contribute to increase the number of internet users , but also the demand for higher bandwidth speed. This is expected to continue because support for network connectivity is now a standard feature on consoles as well.

There is a mutual dependency between entertainment software and content. There are many examples of entertainment software based on movies and the opposite, like Chronicles of Riddick: Escape from Butcher Bay. Musicians, celebrities and sport stars benefit from being used in entertainment software. As for the game this may lead to increased sales.

**Output**

Technology which was once developed for the software industry is now used in non-gaming applications and other sectors. A suitable example is the CELL processor. It was developed by Sony, Toshiba and IBM (STI), and was first used in the PS3 console. Toshiba however, used the powerful processor in a slightly different way. At the International Consumer Electronics Show (CES) in 2008 Toshiba showed what happens when you put a CELL processor in a TV. Among many spectacular features was real-time HD upscaling of standard definition[37]. Other applicable sectors for technology transfer are health care, pollution control, real estate, intelligence testing, manufacturing quality control and military training. As an example, the American army uses the CELL based Mercury computer BladeCenter to handle sonar and radar computation[38].

## 2.5 The Game

Team Fortress 2 (TF2) is sequel to TF which was released back in August 24, 1996. The game were made by Walker, Cook and Caughley as a class-based multiplayer modification (also referred to as a mod) for Quake. A class-based game implies that a player does no longer have the same capabilities as every other player in the game. The game features nine characters, each with their own personality, tactical ability and special weapons. These are scout, soldier, pyro, demoman, heavy, engineer, medic, sniper and spy. This allows players to adopt characters which suit their playing style. The game gained fast popularity among gamers which encouraged them to work on a new game named TF2. The game was never finished. Instead Cook and Walker were hired by Valve Corporation, and three years later TF Classic was released. The game was developed by using the public available HL: Standard Software Development Kit[39]. Finally, in October 10, 2007[2]., TF2 was officially released through Valve's content delivery platform called Steam. The game was a part of a bundle, called "The Orange Box".

### 2.5.1 Getting Started

To play TF2 require first of all a Steam account, which is free of charge. The only thing that costs is purchasing the game. There are various ways to join a server, either through external websites like game-monitor.com and gamespyarcade.com, the downloadable tool HLSW[10] or by using Steam's built-in server browser (figure 2.3).

The browser allows steam users to filter the master server list (contains all servers connected to the Steam network) before connecting. The user can filter search based on location (Asia, Europe, Africa etc.) , anti cheat, latency (RTT), map, if the server has users playing or the server is not full. These options comes in handy as there are TF2 servers by the thousand. According to game-monitor.com[40], a site that

[2]Beta version available September 17 for those who purchased "The Orange Box"

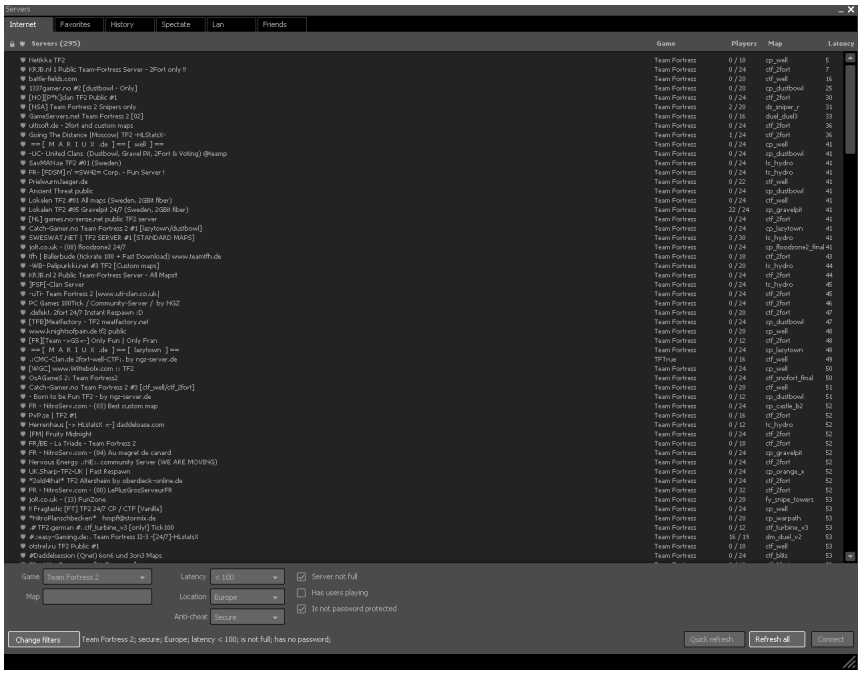



Figure 2.3: The server browser in Steam.

constantly monitors game servers there were 3618 servers available in 07.02.08. This number will vary as servers go down.

### 2.5.2 The Gameplay

The gameplay of TF2 follows the same steps as its predecessor which is class-based, multiplayer warfare on a map. It consists of two teams, Reliable Excavation Demolition (RED) and Builders League United (BLU), which compete against each other (figure 2.4). The game supports twenty four players simultaneously on a game server.

The objective of the game depends of the game mode . Till now, there exist four modes; capture the flag (CTF), control point (CP), territorial control (TC) and payload (PL).

1. Capture the Flag: Involves capturing the enemy's intelligence briefcase which in this case is the flag, and returning it to your base. First team to accomplish three captures wins. Maps from this category are 2fort and Well.

2. Control Point: The goal of the RED team is to defend the control points from attacking team BLU. To capture the opposite team's control point one has to stand on the point for a given time. Maps which fell under this category are Granary, Well, Dustbowl, Gravelpit and Badlands.

3. Territorial Control: The map is split into small territories with a control point on each territory, and the first team to reach final territory wins. Today there is only one TC map, called Hydro.


Figure 2.4: Featuring the character Demoman at RED team on the map named cp_dustbowl.

4. Payload: Shares similarities with the CP mode, but instead of fighting yourself through the map, a bomb payload is pushed through control points. There exist only one map in this category, called Gold Rush.

### 2.5.3 The Architecture

There are two types of network architectures that should be mentioned in the context of online games, Peer-to-Peer (P2P) and Client-Server. In a P2P architecture, the game is designed to utilize the CPU and RAM of the connected peers in order to manage the world state. A peer functions as both a ”server” and ”client”. Using this type of architecture in games has not been common yet. Nevertheless, there have been some research in supporting simple MMGs games on a P2P architecture[41]. In games based on the client-server architecture, there are both clients and server. Team Fortress 2 is based on the Source Engine which utilize this architecture. According to MSDN[42], client-server scales better than P2P and the topology is essential for massive multiplayer online games (MMOG).

Valve describes[43] the server in a client-server architecture as:

> *”(...) a dedicated host that runs the game and is authoritative about world simulation, game rules, and player input processing.”*

As the figure illustrates (figure 2.5), the clients are only connected to the server and the communication goes back and forth between client and the server. Not with other

clients as what a P2P game would do. Communication happens through UDP packets, 20 to 30 per second according to Valve[43]. To avoid bandwidth congestion by sending packet updates whenever something changes in the world, the server takes snapshots at a constant rate of the current world. These snapshots are then broadcasted to its clients.

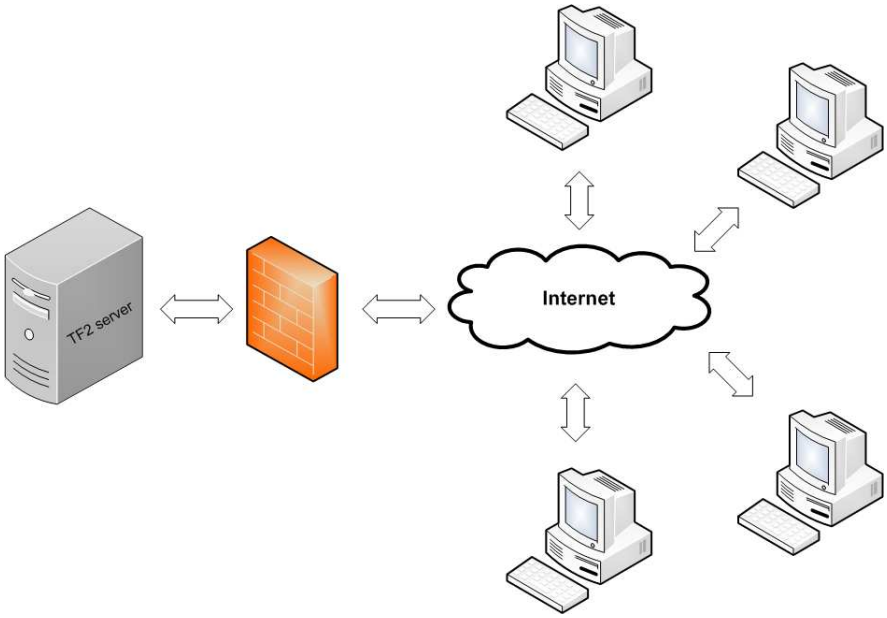



Figure 2.5: Illustrates a client-server architecture

As the server provide world simulation, the clients which are connected act as windows for viewing it. If one of the players in the game moves, then the client which moved has to notify the server in order to affect the world. The server in turn informs all the other players on the server that there has been a change in world state[42]. Based on received world state update from the server, the client generates audio and video output. The clients are also responsible for sampling data from input devices like keyboard, microphone and mouse and send it back to the server for additional processing[43].

The time a packet uses on traveling from the server to the client and back is known as RTT. If the RTT is high or packet loss is occurring during a game, the clients will experience a non smooth gameplay (referred to as lag) where hitting other players is difficult. To cope with such problems Valve[43] uses data compression, interpolation, lag compensation and prediction to make lag less noticeable to the player.

The game server in the figure above (figure 2.5) could either be a listen- or a dedicated server. A listen server runs on the same machine as the player. This means that when the host player decides to disconnect from the server, the server is shut down. The benefit of running a listen server is that it is free because you don't need to buy hardware and software for a new machine. The drawbacks are limited player capacity due to bandwidth, CPU[44] and availability. On the other hand, all this depends on the requirements of the game running. "Game Server Providers" (GSP) takes the dedicated server approach. The server runs on a separate machine and supports more players due

to the CPU does not have to share its system resources as a listen server. Other benefits are that the server can run 24/7 and allows a more fine-grained customization to suite one's needs.

## 2.6 Related Work

There has been conducted some research in the sphere of game servers. However most of these studies are related to traffic analysis and resource provisioning[6]. Examples are studies by Choi et al.[45] which carried out measurements on a MMORPG game called "Lineage II" to characterize the MMORPG traffic, Breu[18] who studied network characteristics of three Counter-Strike servers running on the same computer and Chang et al.[46] which analyzed traffic from multiple FPS game servers.

According to Abdelkhalek et al.[7] there has not been much research on the behavior and scalability of commercial applications like multiplayer game servers, compared to scalability of scientific workload. In their paper they chose a FPS game called QuakeWorld[3] for studying game server scalability and behavior. In order to do this they had to develop a benchmarking methodology. This was challenging for them[7] because:

> *"(i) There is no well-defined input to use for system benchmarking. (ii) The input stimulus is external to the application server (triggered by client systems). (iii) Typical setups require interaction of human users. (iv) The levels of scalability to be studied exceed the size of most university-level laboratories requiring hundreds or thousands of clients."*

The methodology they proposed automated the benchmarking process, compared results and allowed a large scale experiments on a small setup. The automation was made possible using automated players, except for one human player who was always present during the experiments in order to compare his result with the rest of the automated players. The automation of the players was made possible using a recorded event of a demo session.

The experimental testbed consisted of 32 Pentium II, 400MHz computers with dual processors running Windows NT on a 100Mbit private network. They performed two experiments; the first with 1, 2, 4, 8, 12 and 16 players to identify trends for which the server is not degraded. In the next experiment they simulated 1, 2, 4, 8, 16, 32, 64, 80 and 96 players on each client in order to study server performance. Each test lasted for 2 minutes and were run many times to ensure consistency on both experiments. When it comes to the environmental factors (map) where the game takes place, they had only two maps; one large map with high complexity (layout and many objects) and a smaller second map with high interaction level, but simple when it comes to details. The first experiment was only tested with the small map compared to the second experiment.

[3] Multiplayer version of Quake

Their main findings are that processor cycles are the main bottleneck and is fully utilized when the number of players are high. Network bandwidth is not an issue as there is little information exchanged between client and server. Server utilization increases linearly with players on the server. When it comes to memory they have found that this is not a problem.

# Chapter 3

# Approach

This chapter will present the scientific method and the chosen approach. An in-depth description of the approach and how this was solved is also given.

## 3.1 The Scientific Method

The method is fundamental in any scientific research. It is defined by Tranøy[47] as (my translation):

> *"An approach to generate knowledge or re-examine contentions, which are claimed to be true, valid or tenable."*

He differs between two methods: quantitative and qualitative. Quantitative methods deal with things that are measurable. Therefore, the data takes the shape of numbers. This method deals with hypothesis, which are based on the problem statements. Hypothesis commonly suggest a possible correlation between key variables in the problem statement. Before the hypothesis can be answered, the data are often "treated" in advance with some statistically analyzing methods. While quantitative methods investigate the relationship of phenomenons, qualitative methods aim to provide in-depth insight of the phenomenon. Also, research techniques from the latter method give data in terms of text.

There are two ways of approaching a problem, known as deductive- (from theory to empiricism) and inductive (empiricism to theory) reasoning. In deductive, theory is narrowed down into one or more hypothesizes. The aim of this approach is to either improve existing theory or reject it. Inductive reasoning is known as the "opposite" of deductive. This approach is characterized by a vague problem statement[48] where the main goal is to get an increased understanding of the studied phenomenon. Therefore, the data collection method is not decided in the initial research phase, but along the way. This can lead to more exploration in order to gain understanding. Deductive reasoning is commonly used with quantitative research, while inductive in qualitative. However, these two approaches can be applied with both the mentioned methods[49].

In this thesis i have chosen an inductive approach. This demand that the researcher is open minded. To answer the problem statement:

*How can one make a monitoring system for a game server?*

the author must begin with installing a real life service. When the server has been confirmed working properly, ”identity and behavior” has planned to be added through configuration. Identity in the way that the server will be given a unique name (title) and behavior by specify various game settings. After the game server has been set up correctly, the author plans to take actions to make the server popular. It is important that there are players on the server as the future monitoring tool (irrespective of its state) will be used to capture real life data. This data will be used for comparison later on.

The next step will be to begin with the development of the monitoring tool. This requires that the author choose a programming language. In this case Perl is planned to make up the basis of the tool. The author does not have a clear definite conception of how this tool is going to be, which constituent parts, variables and functions which are important for proper monitoring. This uncertainty is expected to make the author do continuously decisions throughout the development, which again will lead to frequently modifications of the tool. Also, Munin is planned to be installed on the machine, as a comparison.

To address the usefulness of the variables, a controlled experiment with real players is planned to be held at school. This experiment will be a unique opportunity to gather user impressions of the running server instance, which in real life would be hard to obtain. Therefore, it is planned to make a quantitative questionnaire which each player has to complete. It is uncertain how to get clients, software (TF2) and players for the experiment. Regarding the software, the author plans to take contact with Valve by mail. As the experiment is assumed to be held at school, the author will ask Oslo University College (OUC) for permission to use twenty-four clients. Concerning the players, the author view students from OUC as potential participants. The selection of participants will not involve sampling techniques like SRS (simple random sampling, a probability sample method). Instead the author plan to use purposive sampling, a non probability sampling technique. We seek ”gamers” as a group and assume that they are to be found among computer students at OUC. This sampling is useful when the targeted group must be reached quickly and generalization is not that important[50].

Both monitoring tool and the questionnaire will give data in terms of quantitative values. Therefore, the author plan to use quantitative techniques for interpreting. It is important to keep in mind that data from a measurement is never 100% correct. Therefore uncertainty will always be an issue. Uncertainty are caused by errors. Mark Burgess[51] differs between personal, random and systematic error. Random error occur randomly and cannot be fully eliminated. As they are unpredictable, the observed value in the experiment can change in both directions ($\pm$ value). The likelihood of the value going $\pm$ are assumed to be equal, and do not have much influence on the averaged value of the measured data. Systematic error cause deviation throughout all

measured data. Compared to random error, the change of value can either go + or -[51]. This type of error is likely to affect the average value of the measured data. Personal error or human error is caused by the person carrying out the measurements without knowing. For instance making wrong calculations.

## 3.2 Set Up A Real Life Service

It was decided to use Team Fortress 2 as a test case for the server. The game has been released recently and was expected to be computationally demanding and also to attract a lot of players.

In order to monitor the performance of a novel service, in this case a game server, a TF2 dedicated server was installed (appendix A) on the provided computer. Before the server could ”go public”, a request for opening port 27015 on IP 128.39.74.31 was sent to OUC. By carrying out the installation steps in appendix A, the server will have default settings only. To customize the behavior of the game server, a configuration file (named server.cfg) was made (appendix B.1). The configuration file for this server is based on Muppet's[52] own ”server.cfg”, but modified based on experience, in-game testing as well as feedback from players on the server. Also, in-game testing in the initial phase revealed a serious performance issue[53]. The game froze out of the sudden and the CPU utilization rose above 90%. Even though this happened for a very short period, players left the server immediately as the game experience become poor (unable to aim and barely move). A possible explanation to this phenomenon might be that the CPU got busy doing something else than processing player input. However, this problem has been solved by upgrading the kernel to 2.6.24-5-server [53] (appendix C.1).

### 3.2.1 Make the Server Attractive

Everyone can set up a game server, but not every game server set up becomes popular. This is a challenge. It is important to attract players to a server, especially if the server is rented. In this thesis it was important because an attractive server will generate more interesting performance data (compared to a server with few or none players). The data will be collected by the the monitoring tool. The items below summarize actions taken by the author.

#### Friends and Clan Mates

As mentioned before, the Steam server browser allows users to filter out servers with none players to reduce the amount of servers appearing in the browser. To get the server ”visible” for others, friends and fellow clan mates were the invited to join the empty server to attract other players. Once the ground is set, it did not take long before the server was full. This was done on a regular basis until other players started joining the server by themselves.

### Statistics

In the initial phase, the server's name was ”Team fortress 2 @ HiO [NORWAY]”. After a period of few players joining the server the author decided to provide ”real time statistics” to increase the number of players on the server. HLstatsX[9] is one of the tools available for game servers today (page 3). It's insufficient in the context of monitoring the performance, but often used as it provides extensive statistics gathered from the game server's logs. Gamers (people who play games) enjoy statistics about their game play. Statistics may contribute to increase the number of players on a server and the possibility of players returning to ”climb the ranks”. The installation of HLstatsX requires some configuring depending of the version. There are two versions, the downloadable free of charge and the Premium which cost 15€for three months. The first mentioned require installation and configuration of MySQL server, PHP and Apache server (appendix C.2). With the Premium version everything is set up at HLstatsX own servers and the process of making statistics work with the game server is simple (appendix B.4). The reason for choosing the pay version instead of setting up HLstatsX on the same server was to focus on accurate measurements. As the server now provided stats, the name of the server was changed to ”Team Fortress 2 @ HiO [NORWAY] - HLstatsX enabled”.

### Server Title

Each day the number of TF2 servers increases and the competition to attract players connecting to a server grows. It's important to stand out because who wants to pay a monthly fee for an empty server? A catchy server name (to attract attention) with a pinpoint description of the server is crucial for success. Players are very demanding, so they know what they want and what to look for when browsing the master server list. Based on this reflection, the name was changed once again to ”HiO | All welcome | No lag | Dustbowl/Badlands | HLstatsX”. The title tells a user that this server is OUC's (HiO) property, lag will not appear on the server, dustbowl and badlands is the current map rotation and that real-time statistics is currently running.

### Map

The server uses only two maps[4] all the time. The map cp_dustbowl remains fixed all time, while the other changes now and then. To inform others about this change, the server title must be changed as well.

### Events

The author took advantage of anticipated updates for the game, like new maps or other features. The server was updated almost immediately after the release and the change was informed to others through the server title.

[4]The experiment was run with the map cp_dustbowl and cp_badlands.

## 3.3 A Monitoring Framework

Monitoring tools made for game server are per date limited (introduction, page 3). It is assumed that the lack of proper tools leads people to develop their own custom tools in shape of scripts. These scripts collect data which they consider important in order to keep track of the performance.

The monitoring tool in this thesis consist of three components: (1.) collecting data (values), (2.) analyzing data and (3) update visualization of data. This process is described in section 3.3.1 and 3.3.2 respectively.

Figure 3.1 is a attempt to illustrate how the tool developed in this thesis works individually and together.

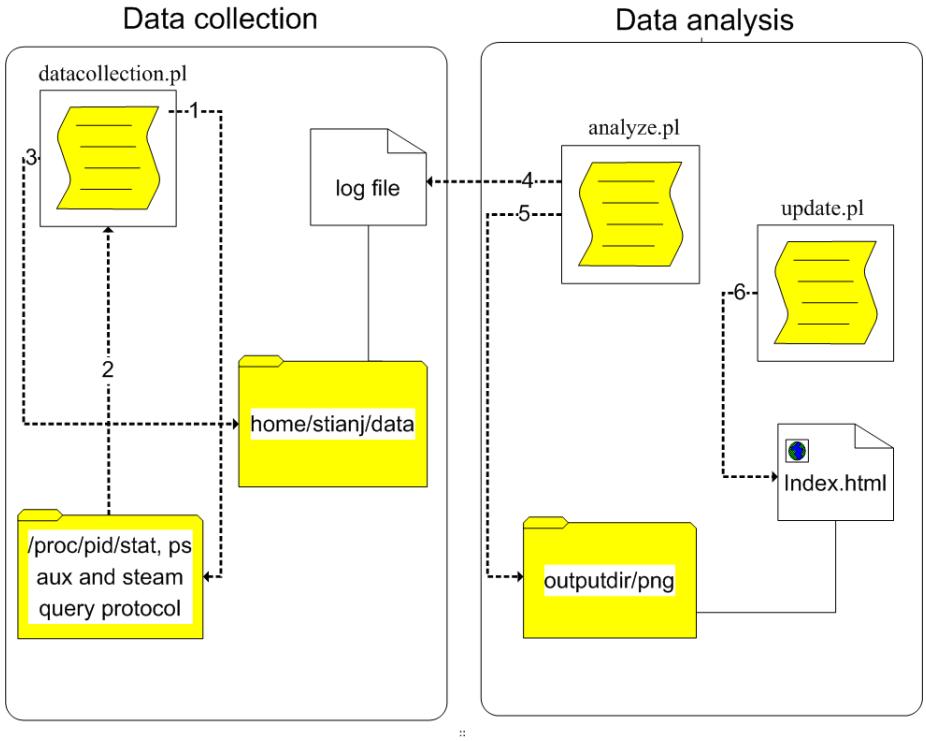



Figure 3.1: Illustrates the process of collecting and analyzing data

Each number represents an action in the monitoring tool. In (1) data are collected (constantly every twenty seconds) from three separate places. These values are (2) ”sent back” to the script which then (3) stores them in a log file located in the home-/stianj/data folder. To (4) visualize the data, a new script takes over. It (5) fetches the given log file and generate a web page with figures for each of the collected variable. As we want to keep track of the performance, (6) the figures are kept up to date by a third script.

The three components are written in Perl (version 5.8.8). Perl is easy to adopt to those who are not experienced in programming. Perl is the abbreviation for ”Practical Extraction and Report Language”. It is a open source interpreted language which

support many platforms today, but was original designed for UNIX OS. The language itself is a mixture of shell programming, UNIX utilities (e.g. grep and sed) and C features. Perl is a popular language among system administrators because it makes manipulation of processes and files easy[54].

### 3.3.1 Online Data Collection

#### Identify Relevant Variables

A lot of time was dedicated to the process of identifying key variables to be measured. It was an important part of the process of developing the data collection script. All variables that are collected are described briefly in tables below.

Epoch is used to keep track of when the data is collected. It provides the most detailed time stamp in UNIX as it includes seconds and more, not hh:mm only.

| # | Name | Description |
|---|---|---|
| 1 | Epoch | The present month/day/year and the time (hh:mm:ss) is represented as seconds since epoch. The epoch started at January 1 1970 00:00:00 GMT, which is in fact equal to 0 seconds. |

Table 3.1: Time variable

Monitoring the performance of a game server requires variables that can tell us something about the performance of the process. A obvious place to begin is the ps command. The command gives a short overall report about the process's current state. We collect four variables from ps (table 3.2).

| # | Name | Description |
|---|---|---|
| 2 | START | The time when the ps command was executed. |
| 3 | PID | An abbreviation for Process Identification Number. A unique number which each running program in Unix are identified by. |
| 4 | %CPU | The Central Processing Unit (CPU) executes processes. %CPU is the used CPU time divided by the current running time of the process. |
| 5 | %MEM | The ratio between the process's Resident Set Size (RSS) and the physical memory (total memory installed) of the server. |

Table 3.2: Process Status variables

However, these variables provide only the basics. To go further into the working of a process, we move on to proc file system. The subdirectories located under /proc/ allows us to look into parts of the kernel's data structures[15]. Many variables are collected from /proc/pid/stat. It is a subdirectory which gives status information about the process. The first eight variables (various pid's, filename of the executable, its state, session ID and process group ID) are dropped. The rest are collected as we do

not know at this point which variables that are interesting. The name and description in table 3.3 is obtained directly from the proc manual (man proc) on the server.

| # | **Name** | **Man Description** |
|---|---|---|
| 6 | Flags | The kernel flags word of the process. |
| 7 | Minflt | The number of minor faults the process has made which have not required loading a memory page from disk. |
| 8 | Cminflt | The number of minor faults that the processs waited-for children have made. |
| 9 | Majflt | The number of major faults the process has made which have required loading a memory page from disk. |
| 10 | Cmajflt | The number of major faults that the processs waited-for children have made. |
| 11 | Utime | The number of jiffies that this process has been scheduled in user mode. |
| 12 | Stime | The number of jiffies that this process has been scheduled in kernel mode. |
| 13 | Cutime | The number of jiffies that this processs waited-for children have been scheduled in user mode. |
| 14 | Cstime | The number of jiffies that this processs waited-for children have been scheduled in kernel mode. |
| 15 | Priority | The standard nice value, plus fifteen. The value is never negative in the kernel. |
| 16 | Nice | The nice value ranges from 19 (nicest) to -19 (not nice to others). |
| 17 | 0 | This value is hard coded to 0 as a placeholder for a removed field. |
| 18 | Itrealvalue | The time in jiffies before the next SIGALRM is sent to the process due to an interval timer. |
| 19 | Starttime | The time in jiffies the process started after system boot. |
| 20 | Vsize | Virtual memory size in bytes. |
| 21 | Rss | Resident Set Size: number of pages the process has in real memory, minus 3 for administrative purposes. This is just the pages which count towards text, data, or stack space. This does not include pages which have not been demand-loaded in, or which are swapped out. |
| 22 | Rlim | Current limit in bytes on the rss of the process (usually 4294967295 on i386). |
| 23 | Startcode | The address above which program text can run. |
| 24 | Endcode | The address below which program text can run. |
| 25 | Startstack | The address of the start of the stack. |

| | | |
|---|---|---|
| 26 | Kstkesp | The current value of esp (stack pointer), as found in the kernel stack page for the process. |
| 27 | Kstkeip | The current EIP (instruction pointer). |
| 28 | Signal | The bitmap of pending signals. |
| 29 | Blocked | The bitmap of blocked signals. |
| 30 | Sigignored | The bitmap of ignored signals. |
| 31 | Sigcatch | The bitmap of caught signals. |
| 32 | Wchan | This is the ”channel” in which the process is waiting. It is the address of a system call, and can be looked up in a name list if you need a textual name. |
| 33 | Nswap | Number of pages swapped (not maintained). |
| 34 | Cnswap | Cumulative nswap for child processes (not maintained). |
| 35 | Exitsignal | Signal to be sent to parent when we die. |
| 36 | Processor | CPU number last executed on. |
| 37 | Rt_priority | Real-time scheduling priority. |
| 38 | Policy | Scheduling policy. |
| 39 | Delayacct_blkio_ticks | Aggregated block I/O delays (measured in clock ticks (centiseconds)). |
| 40 | No description found | No description found |
| 41 | No description found | No description found |

Table 3.3: Proc variables

Collecting data about the process is important, but so is in-game data like current number of players on the server and map. As mentioned before, HLstatsX generate statistics and graphs by using the log files which the game server produce during its uptime. Integrating HLstatsX information like e.g. number of players to a script is not possible in this case as the log files are forwarded directly to a non accessible server. But, this can be overcome by disabling the log forwarding. However, processing the log file line by line in a script may not be acceptable as the log file increases rapidly in size over time. The most suitable approach is to use Steam's own protocol[14] for querying Steam game servers. The server responds to four different queries[14]:

1. A2A_PING: Check if the server is alive.
2. A2S_INFO: Retrieve summary information about the server.
3. A2S_PLAYER: Give details about each player currently on the server
4. A2S_RULES: Provide information about the server rules.

Querying in-game information from a server is done by sending UDP packets. Each query are approached in different ways. This protocol is used by game-monitor.com among others to provide extensive information from game servers running different games[14]. The A2S_INFO query is in this case the most useful query. From here three variables are collected (table 3.4).

| Nr | Name | Description |
|---|---|---|
| 42 | Map number | Current map of the server. The value 0 implies that the current running map is cp_badlands, 1 is for cp_dustbowl. |
| 43 | Players | Current number of players. |
| 44 | Max Players | How many players that are allowed to play simultaneously. |

Table 3.4: Game variables

### Collection Script

The collection script was originally designed to start a game server in screen and then begin logging data. Screen is a command that allows a user to create multiple virtual terminals in a single terminal window. This is beneficial for many reasons, like keeping processes running even though one logs out or disconnects (SSH) from the machine. The script took two arguments. (1) name of the screen session and (2) port number of the game server. Both arguments had to be there in order to run the script. Automatically logging of data each time the game server starts is not desirable in some cases like testing changes made to the server configuration or new updates. Therefore it was modified. Now the script is executed with the following options.:

```
Usage: [ -P <PID>] [ -I <IP>] [ -L <LOGFILE>] [ -S <SCREEN-SESSION>] [ -p
<PORT-NUMBER>]
```

The IP of the server is the only option which is mandatory of those presented. Without it the script refuses to run. If PORT-NUMBER is not specified on run the script uses port 27015 as default. The PID is only specified in scenarios where the game server process is already running. To collect data in these cases, one has to specify the running game server's PID along with the LOGFILE option. Note that the extension of the file must be specified manually in this case. The benefit of this approach is that we do not have to kill the server every time we want to check the performance. The SCREEN option is the name of the screen where the server will be run.

In both cases, the script slept (waited) for one second before the data gathering process starts. Because the dedicated server process (srcds_i486) might not have started. The script begins by collecting variables from the ps command. These are stored in an array. The PID from the ps is then used to collect additional data by opening the /proc/PID/stat file.

The process of collecting game variables was a bit tricky. To begin with, the script established a socket connecting to the game server, and died if a connection cannot be made, as shown in figure 3.2.

If successful, the script sends a UDP packet containing predefined byte values over the socket. The server automatically respond by sending back a response packet back if the first packet had the correct byte values. If we receive a response from the socket, the packet has to be unpacked properly. If not we would get a string representing of the structure like the one below.

```
1
2  my $socket = IO::Socket::INET->new(
3                                  Proto=>"udp",
4                                  PeerPort=> $opt{p},
5                                  PeerAddr=> $opt{I}
6                                   )
7    or die "Can't make UDP socket: $";
```

Figure 3.2: Establishing a socket connection to game server

IHiO | All welcome | No lag | Dustbowl/Badlands | HLstatsXcp_dustbowltfTeam Fortressdl1.0.2.0i

Therefore, defining how it should unpack the received packet is crucial to parse out all ”hidden” information (values) correctly. To get the true information we need to tell the unpack() function how it should treat the packet. This is done by setting characters matching the description in the ”Reply format” table[14].

```
1  if ( $respons ){
2          ($a,$type,$version,$hostname,$map,$gamedir,$gamedesc,$appid,
3           $players,$maxplayers,$bots,$dedicated,$os,$password,$secure,
4           $gameversion) = unpack("iACZ*Z*Z*Z*sCCCaaCCZ*",$respons);
```

Figure 3.3: Unpack response packet.

When the response is received and unpacked the socket is closed down. The process of querying the server is included as a subroutine in the script. For the change to take effect, the server was started with a new log file (proc19).

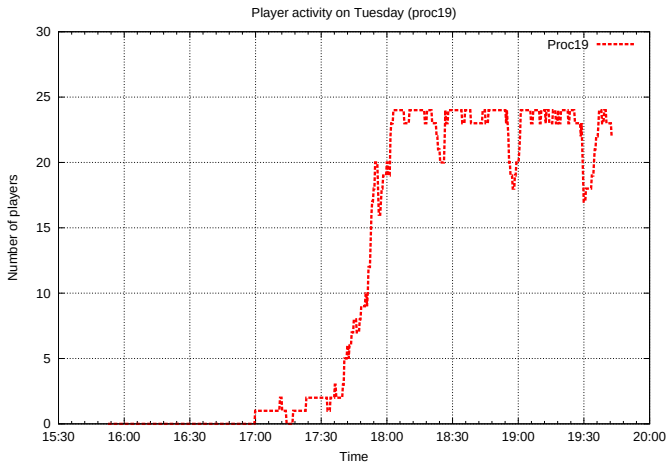



Figure 3.4: Shows player activity on Tuesday.

The number of players connected to the server increases considerably after 5:30 PM (figure 3.4). The downward spikes are players leaving the server. The figure shows that

the script ran fine for almost four hours before it stopped logging. The first reasonable explanation that comes to mind at that point, is ”server is shut down again[5]”, but this was not the case this time. The next day the server was executed with a new log file (proc20).

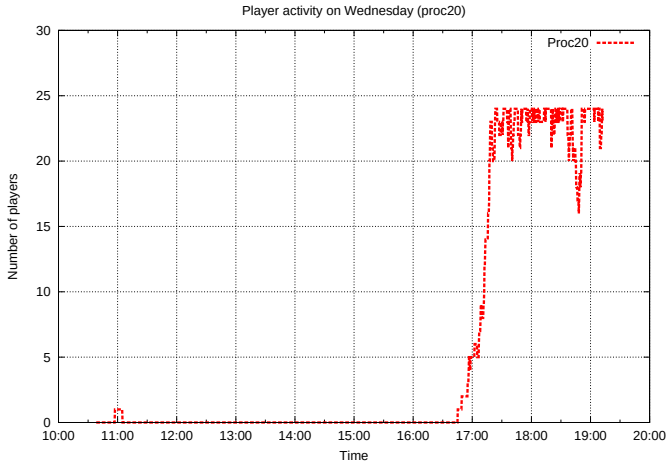



Figure 3.5: Shows player activity on Wednesday.

Like figure 3.4, the server becomes ”active” around 5:00 PM. The number of players varies between twenty and twenty-four, except for a drop around 6:50 PM. Compared to proc19, it ran more than seven hours before it stopped (figure 3.5). Both scripts stopped at a point where there were a lot of activity on the game server. According to OUC’s own HLstatsX web page there were a lot of player activity on on Tuesday and Wednesday after the script had stopped (figure 3.6).

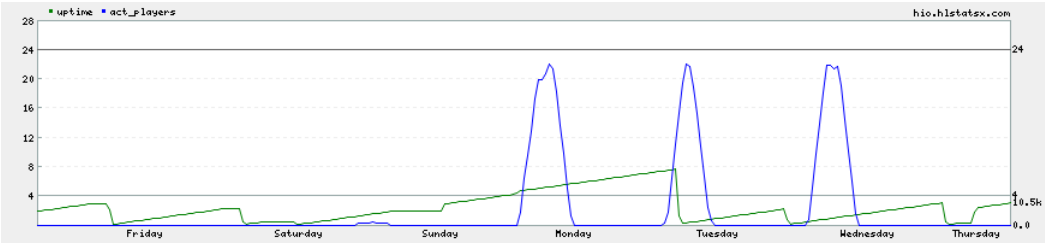



Figure 3.6: Player activity on the game server, week 10th. The figure shows a a similar activity pattern for Monday, Tuesday and Wednesday.

The only logical place where the script could hang was if recv() function did not receive the expected packet response from the socket and thereby waiting endlessly. As the number of players gets high, the server works hard to keep everyone up to date about the world simulation. Sooner or later packet loss will occur. To prevent this from stopping the script now and then, the subroutine was modified with an alarm timeout (figure 3.7).

[5]The game server was shut down frequently by someone throughout the semester.

```
1       $TIMEOUT = 4;
2    eval {
3        local $SIG{ALRM} = sub { die "alarm time out" };
4        alarm $TIMEOUT;
5        $socket->recv($respons, 1400);
6        alarm 0;
7        1;
8    } or $respons = "";
```

Figure 3.7: If the response packet is not received after four seconds, fixed values are set.

If no response packet is fetched, then the values -1 -1 -1 are returned to the log file. An example of a line from the log file looks like this:

```
1202480844 15:27 8536 97.0 3.0 4202496 15968 0 0 0 87 11 0 0 20 0 1 0
17995004 114098176 11679 4294967295 134512640 134561892 3214010320 3213996032
3086165008 0 0 0 0 0 0 0 17 0 0 0 0 0 0 0 1 24
```

Every twenty seconds a line of data from the variables is applied in the log file at ”/home/stianj/data”. The author believe that collecting data at this interval is adequate in order to observe the essential changes on the server in the experiment. With a lower resolution (higher time interval) we are more likely to miss out important data. Also, collecting data too often may affect the performance of the system.

### 3.3.2 Offline Data Analysis

The log file from the collecting process consist of 44 variables separated in columns. Specifying a plot file for each of these columns is not only cumbersome, but also time consuming. There may be a case where there are more than just one log file that has to be plotted, making things worse. A better and more scalable solution is to integrate Gnuplot (version 4.2) in Perl. This is beneficial for several reasons:

- A plot file loaded in Gnuplot cannot output multiple figures at once, only multiple plots appearing in the same figure by specifying multiple columns in a file (e.g. 1:2, 1:3, 1:4 etc). A script could do this task by running in a loop.
- To plot other data files than the one already specified, the name of the file has to be changed manually or by using the Unix utility ”sed”. Another approach is to take the wanted data file name as a command line option in a script.

The script is designed to take two arguments: (1) the path of the log file and (2) the path to where the outputted figures should be stored.

```
Usage: [<LOGFILE>] [<OUTPUT-DIR>]
```

The script will not run if the required arguments are not present. In the given output directory a subdirectory named ”png” is automatically created. This is where all figures are stored. Instead of defining .plt files for each column we want to output, a set

of commands representative for all columns is created. To output multiple figures we use a for loop since the number of iterations are known on forehand. The body of the loop contains the code which should be executed repeatedly. The code calls Gnuplot, prints some commands, then closes it when the loop has been completed. The x axis and log file is the only variables that remains ”constant” throughout the loop. To differ between the figures all columns are given explanatory titles which are put in a hash. The generated graphs are represented as time series[6]. All graphs are then presented through a web interface (figure 3.8) to get a quick overview of the performance. The web page which displays the graphs is generated in the same script.

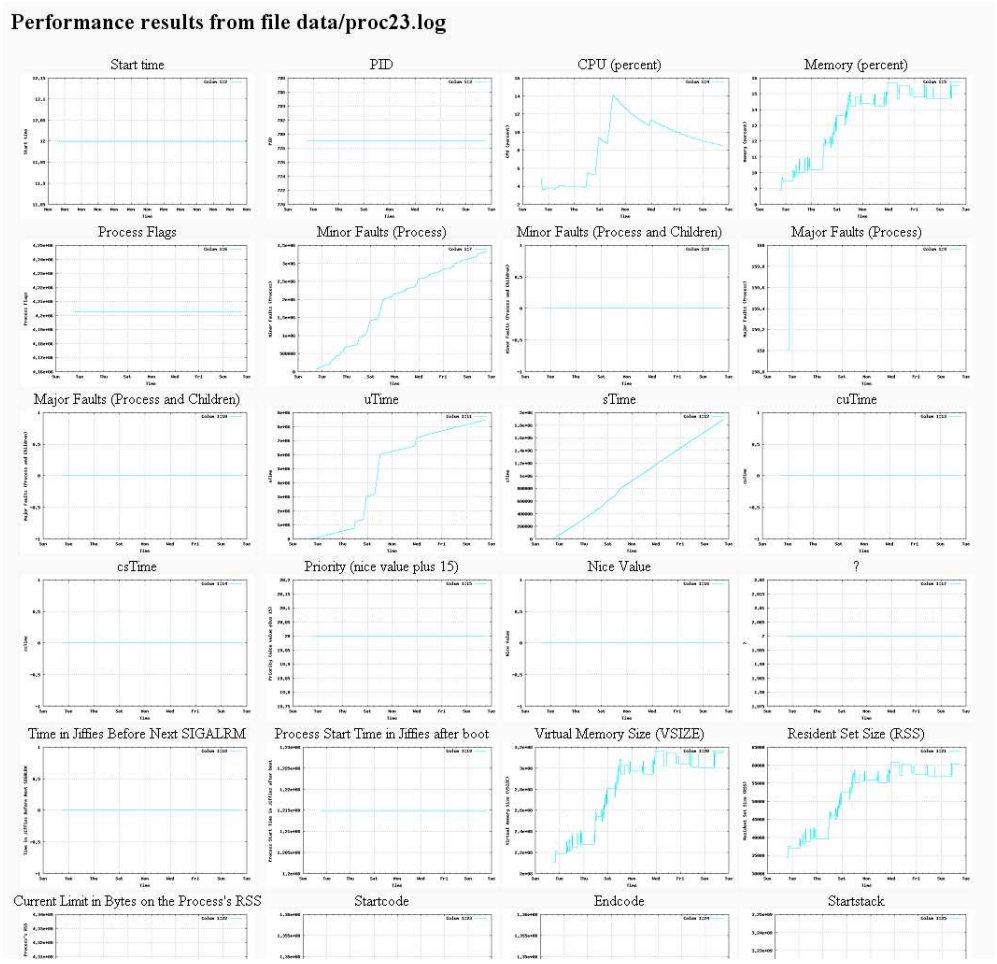



Figure 3.8: Example of an auto generated web page for the proc23 log file

### Update Script

To see the graphs change on the web page as the log file grows, an additional script is made.

[6] A measured value collected at a regular interval is plotted against the time.

```
Usage: [<LOGFILE>] [<INTERVAL>]
```

The script takes two arguments: (1) the name of the log file and (2) how often the graph should be updated. What the script actually does is executing the script created above. The code which updates the graphs runs in a infinite loop. The time interval argument specifies how long the script should sleep before continuing the loop.

## 3.4 Controlled Experiment

### 3.4.1 Hardware Equipment and Software

OUC provided the machine (table 3.5) which the game server was installed on.

| The Dell Desktop PC | |
|---|---|
| CPU | Intel® Pentium® 4 CPU 2.80GHz, 1MB cache |
| RAM | 1536MB |
| Hard Drive | 150GB SATA |
| Operating System | Ubuntu 7.10 (Gutsy Gibbon) |
| Kernel version | Now 2.6.24-5-server (buildd@vernadsky)<br>Before 2.6.22 generic |

Table 3.5: Specifications of the server

They also provided clients (table 3.6) to each participant.

| The Dell OptiPlex 745 clients | |
|---|---|
| CPU | Intel® Core™ 2 CPU 6600 @ 2.40GHz (2 CPUs) |
| RAM | 2046MB |
| Operating System | Windows XP Professional, Service Pack 2 |
| Card Name | NVIDIA Quadro FX 550 |

Table 3.6: Specifications of the clients

### 3.4.2 Preparations In Advance

Abdelkhalek et. al[7] simulated the behavior of ”real players” in both of their experiments, except for the one human player. The players in this experiment will not be automated (recorded event), but real human beings playing on each client. Using human players created three problems:

- Every player must have a client that could run TF2.
- Every player must have a Steam account with TF2.
- Get twenty-four voluntary students to participate

To avoid potential problems and ruining the experiment, preparations was taken seriously at an early stage. Fortunately, the first problem was solved quickly thanks to Tore Øfsdahl who gave permission to install Team Fortress 2 on twenty-four clients.

### Contacting Valve

Being an active user of the Steam Platform, the author knew that Valve operated with Steam ”Guest Passes”. A guest pass comes along with the purchase of a certain game, like the Orange Box. Having a guest pass, the users can share a game for a limited period with a friend who does not own the game. The pass could either be sent to this friend’s Steam username or email address. If sent to an email address, the friend is required to create a steam account in order to receive the pass. The game is available for download after the user has logged on the Steam platform. Compared to a traditional demo games, the friend can enjoy all features of the game until the period expires. After the expiring date, the user has to purchase the game through Steam in order to continue playing.

Guest passes for Team Fortress 2 has not been made public available for users who have bought the game so far. Irrespective of this fact, the possibility of getting ”guest passes” for each of the participants was looked into. The author contacted academic licensing at Valve by mail where the experiment and the current situation was explained briefly. Their reply was exclusively positive. However, instead of using guest passes they recommended Temporary Steam Tournament Accounts. According to Valve® these accounts are provided for academic institutions participating in the Valve Academic Licensing Program or SourceU (appendix D.1), but also used in game tournaments and LAN parties. Temporary Steam Tournament Accounts are easier to handle than guest passes as they come with predefined usernames and passwords. This was beneficial in terms of time as Team Fortress 2 could be installed right away on the clients (figure 3.9) after the receiving the account details on email.

Each client is identified with a number. This number corresponds to the Steam user account installed on that particular machine. E.g. client number one was installed with steam user TU0100200PC1 etc.

### Get Participants

In order to delimit the number of recipients, an email was sent to computer students at OUC (appendix D.2). The experiment was announced as a ”game evening with pizza and soda”, but with a technical focus. The clients provided by the school are basically not meant for playing computer games (due to lack of GPU power), therefore the students were given the opportunity to bring along their own desktop/laptop. To ensure that the experiment was carried out regardless of the student response, friends and family was asked too. Fortunately, the author received emails from exactly twenty-four students. However, one student did not show up that day, thus forcing the author to fulfill the role of a participant in addition to instruct the participants during the experiment.

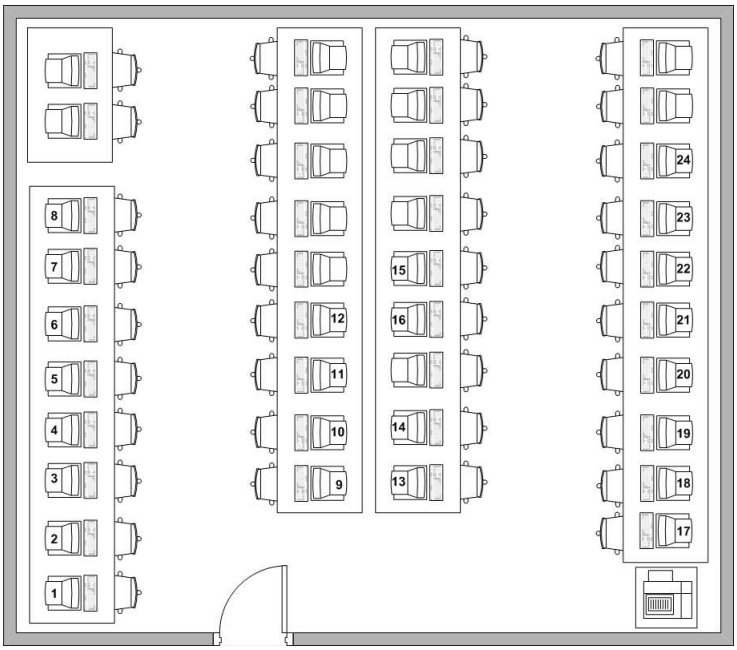



Figure 3.9: Illustrates the classroom where the experiment takes place.

### 3.4.3 Performing The Experiment

The experiment was held on a weekday at OUC from 5:00 p.m. to 9:00 p.m.. Like Abdelkhalek et. al[7], the server had two maps[7]. Carrying out a similar experiment to them would required more time than available in this case. Each test they carried out lasted two minutes. Taking their approach would definitely ruin the game experience as the game would almost end right after it started. Instead, the number of tests were cut down to three scenarios lasting fifty minutes each. Each scenario was believed to put further stress on the system (table 3.7). The player column in table indicates the number of players that should be present on each server in the given scenario.

To organize things, the participants were split into three groups: A,B and C. Client 1-8 was reserved for group A, 9-16 for B and 17-24 for C (figure 3.9). Each students were given a paper (appendix E.1) and a questionnaire (appendix E.2). The paper was made to guide them through technical issues which had to be taken care of before the experiment could begin, but also to giving the users a short introduction about the experiment and guiding them during the experiment (timetable). In addition to the paper, each student were asked to fill out a set of questions after each session. The questions focused on user satisfaction and gameplay feeling. E.g. how smooth did the game run while playing?

To prevent non-invited players from joining the server before and during the experiment, the game servers were in advance set to operate as private game servers. This was accomplished by setting the variable to ”sv_password ”passgohere” ” in the server configuration file. Since the three servers are executed with the same server.cfg file

[7] cp_dustbowl and cp_badlands

| Scenario # | Players Per Server Instance | Description |
|---|---|---|
| 1 | 24 | Running one game server. Group A, B and C joins server on port 27015. |
| 2 | 12 | Running two game servers simultaneously. PC1 till PC12 joins the server on port 27015 and the rest joins the server on port 27016. |
| 3 | 8 | Running three game servers simultaneously. Group A joins server on port 27015, B on 27016 and C on 27017. |

Table 3.7: An brief overview of the execution process.

(appendix B.1) we only need to do this once. Alternative ways to change the variable is to use RCON[8] or use HLSW. In all cases the connecting user gets prompted for the predefined password.

**Automation Script**

The scripts that has been created so far collects, plots data and updates the graphs at a given time interval. Executing the three scripts manually for each of the presented scenarios is a daunting and risky task. There is no room for error in this experiment since it cannot be reproduced later on. That being said, a fourth script was made specifically designed to carry out each scenario. It is different from the others as it is a shell script and does not take arguments from the command line. After the command (Usage: ./execute.sh) is executed the screen is cleared before displaying the following:

```
--------------------------------------------------------------------
Start, runs a team fortress 2 server w/ analyzing & logging.
Stop, terminates everything which was initialized with Start.
Also, the number indicate the number of instances which will
start and stop. E.g. start2 = two game servers are executed.
--------------------------------------------------------------------

Usage: 'basename ./execute.sh' start|stop|start2|stop2|start3|stop3
```

The script uses "case statement", where a value is compared to one or more defined values. When a match is made, the commands in that particular value are executed. In this case there are seven values, start and stop for each scenario and a wildcard character value (*)) that matches everything in case of typos. This was executed on the author's laptop over SSH. It eased the administration considerably during the experiment as the author did not have to execute the individual scripts manually.

[8] E.g. rcon sv_password passgohere.

# Chapter 4

# Results

This chapter presents the results from the three different scenarios. Figures and tables are used to illustrate the findings.

## 4.1 Results From The Questionnaire

To recap, in scenario 1 all players played on the same game server. In the next scenario, two game server instances were run simultaneously on the machine and player per instance were twelve. In scenario 3, three instances were run simultaneously with eight players on each instance.

As mentioned, all participants were asked to fill out a set of questions after each scenario. The main findings from the questionnaire are presented below.

### 4.1.1 Scenario 1

Figure 4.1 shows that 62.5% of the players had an excellent game experience while playing. 58.82% of the experienced players and 71.42% of the non experienced players rated the game as ten on the scale. The average game experience is 9.08.

Also, all players would continue to play on the server (figure 4.2) even if they were not a part the experiment.

### 4.1.2 Scenario 2

Figure 4.3 shows that only one player had a excellent game experience. The largest part (37.50%) of the of the players rated their game experience to 8. All non experienced players are located between 7 and 9, compared to the experienced which have an overall distribution on the scale. Compared to scenario 1, it seems that when one extra server was added the players experienced a drop in user satisfaction. This is confirmed by the average game experience, which is 7.54.

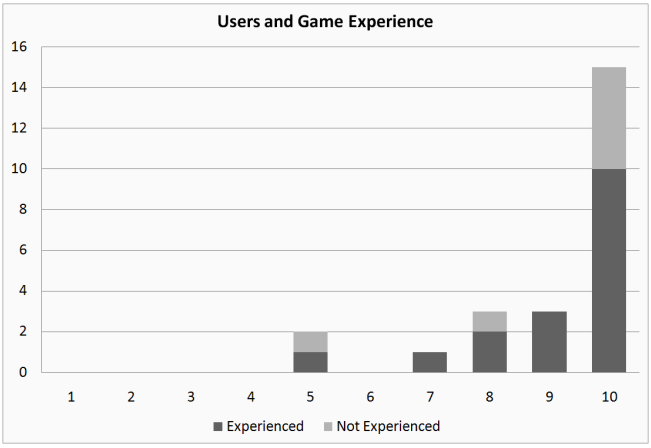



Figure 4.1: How smooth did the game run while playing?

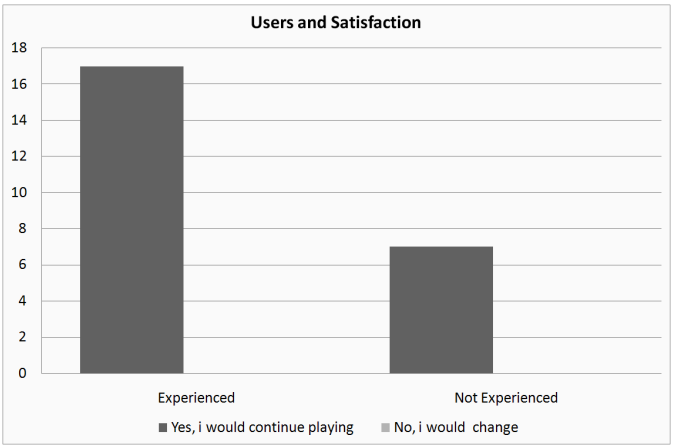



Figure 4.2: If you were playing on this server in leisure time and the playing was not a part of a experiment, would you still continue playing or change to another server?

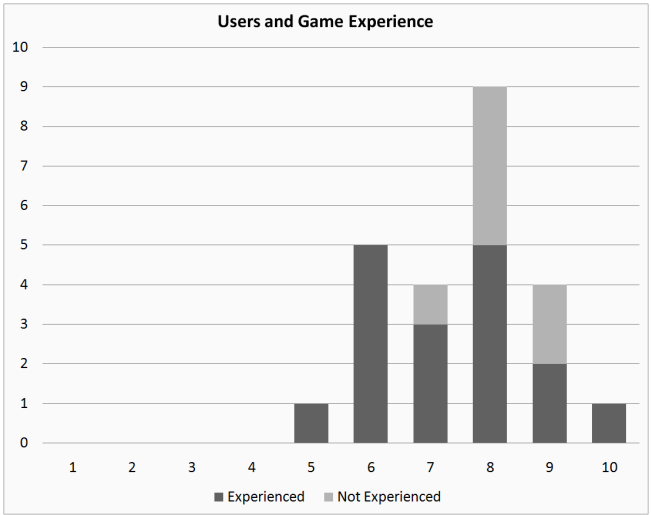



Figure 4.3: How smooth did the game run while playing?

This drop was not satisfactory for one player, which turned out to be a experienced player. If he was not a part of the experiment, he would have left the server (figure 4.4) to play on another game server.

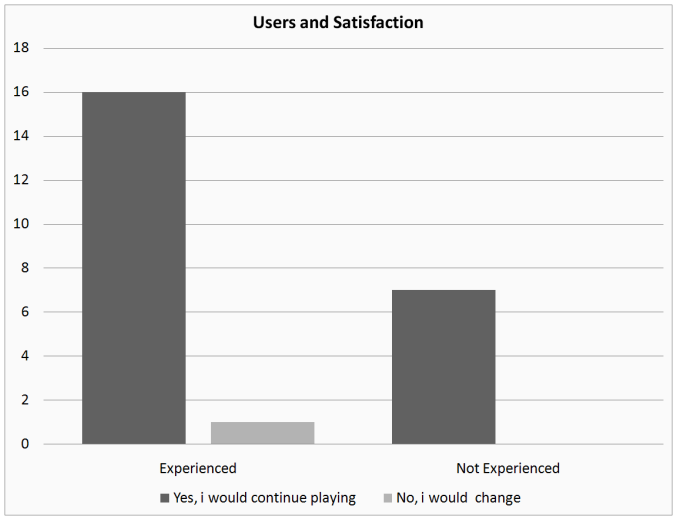



Figure 4.4: If you were playing on this server in leisure time and the playing was not a part of a experiment, would you still continue playing or change to another server?

### 4.1.3 Scenario 3

54.16% of the players lies between 9 and 10 (figure 4.5). Once again we find that the non experienced players are more satisfied with the game experienced than the experienced. 85.71% of the non experienced lies between 7 and 10, compared to 58.33% of the experienced players. The average game experience is 8.04. This tells us that the players are according to their answers more pleased with the game experience than in scenario two.

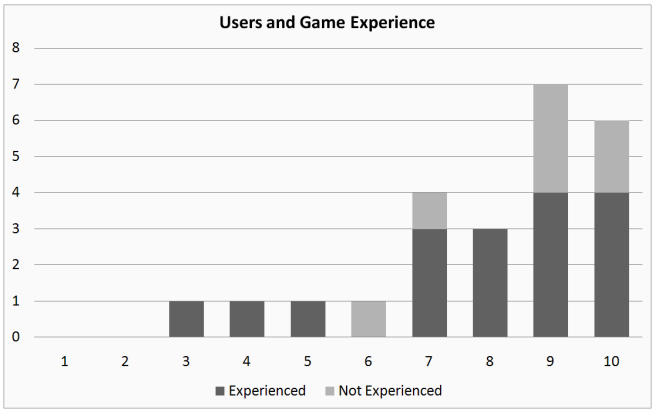



Figure 4.5: How smooth did the game run while playing?

Even though the players were more satisfied in this scenario, two players would have left the server. Both were experienced players (figure 4.6).

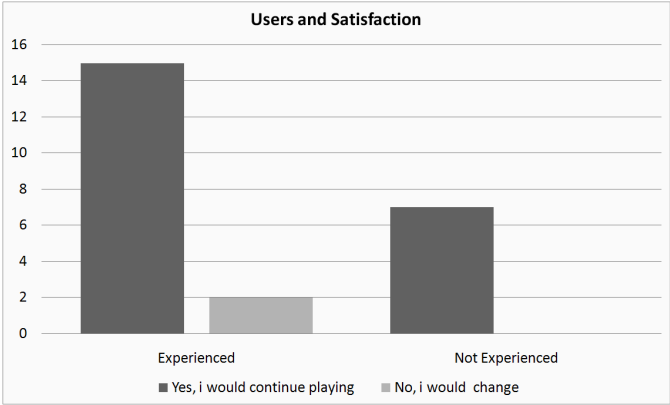



Figure 4.6: If you were playing on this server in leisure time and the playing was not a part of a experiment, would you still continue playing or change to another server?

## 4.2 Results From Experiment

The results from the controlled experiment are based on the output from the developed monitoring framework. Not all results will be presented due to the fact that many variables did not collect any data at all. The most interesting finds from each scenario will be presented in a single graph as it gives a good overview of the differences in the data. To avoid confusion, data from the each server is labeled (table 4.1):

| Scenario | Server instance |
|---|---|
| Scenario 1 | Server1 |
| Scenario 2 | Server2 and Server3 |
| Scenario 3 | Server4, Server5 and Server6 |

Table 4.1: Server instance in each scenario.

E.g. the data from the first game server instance in scenario 1 correspond to server1.

### 4.2.1 Players

Figure 4.7 shows a comparison of the player count on each server.

Compared to the other servers, server1 did not reach the maximum number of players until after a while. The irregularities in the beginning of scenario 1 are caused by participants who logged into other steam account than given in advance. Steam notifies the respective user about this change by (1) not allowing the user to establish a connection to the server or (2) being immediately disconnected from the server while

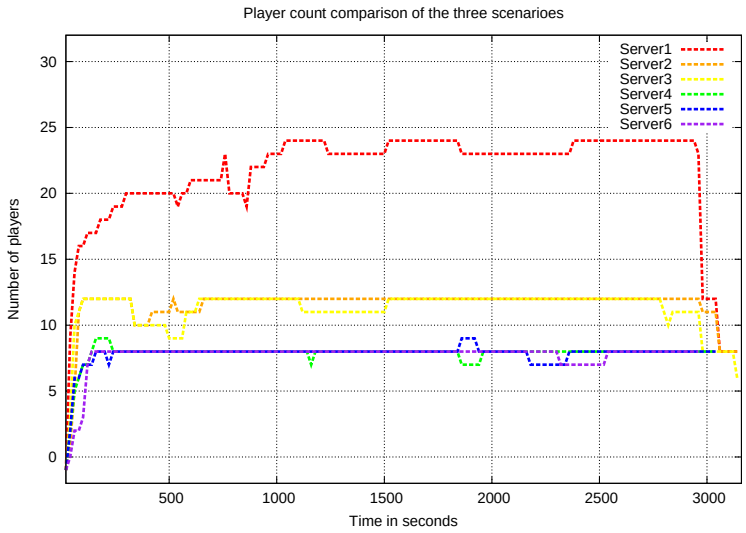



Figure 4.7: Comparing the player count in each scenario.

playing. Before they could log in and join the server again, the author had to find out which user names were available.

The drop in beginning of server2 and 3 (second scenario) was caused by network problems. This resulted in a client timeout for most of the players connected to the school's wireless LAN. On server4 and 5 we see that someone made a connection to the wrong server as the number of players reached nine.

The sudden drops after the graph stabilizes on each instance is assumed to be caused by idle players being automatically moved to spectator mode for a period.

### 4.2.2 CPU%

The Central Processing Unit (CPU) job is to execute a process's instructions. A CPU can operate in two ways, kernel- (stime) and user mode (utime). According to Stallings (2001)[55], software in kernel mode controls the memory, registers, processor and its instructions. The user mode is a non-privileged mode, where the software is not trusted.

Due to the fact that the game server is equipped with a single CPU means that it cannot execute more than one instruction simultaneously. This does not imply that we are bound to run only one program like Word at a time in e.g. Windows. The reason for this is because of multitasking/multiprogramming. This feature can be exemplified with two running programs named A and B. When A has to wait for I/O, the CPU can switch to B which is not waiting, and vise versa. Multitasking leads to higher CPU utilization as it is kept busy.

### CPU% derived from ps

The comparison in figure 4.8 reveals a similar pattern on all servers. The six graphs begins with a high peak before it decreases considerable, then increases to a point where it becomes stable. The high peak is probably a result of the CPU preparing the game server.

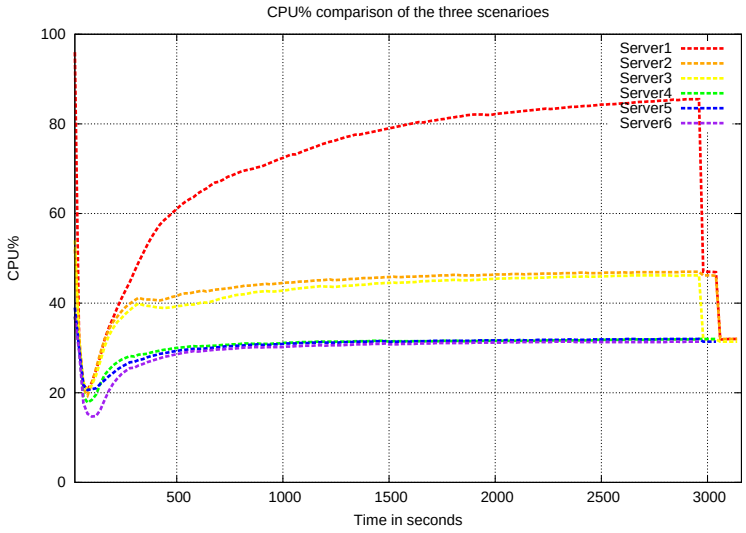



Figure 4.8: Comparing the CPU% in each scenario.

The figure shows that the server1 process utilizes most CPU time during its lifetime. It is also the scenario with the largest player count. The graph also reveals that server2 and 3 uses approximately half of the CPU% compared to the first scenario. The player population on these two servers were the half of scenario one. The last scenario (server4, 5 and 6) utilize least CPU% of all servers, but is also the scenario were the number of players are reduced to eight on each server.

The collected CPU% variable values from the ps utility gives us a smooth graph (figure 4.8). The reason for this is CPU% divides the used CPU time by the current total time of the running process. The graph is useful in the way that it gives a hint about the CPU usage on each server.

### CPU% derived from /proc

To see the fluctuations and the true CPU%[9] usage, new figures (figure 4.9, 4.10 and 4.11) are made.

[9]This was acchieved by adding the values from the stime and utime variable together and then divide by 20. But first we had to find the real growth by calculating the difference of two values due to the fact that we are dealing with cumulative data.

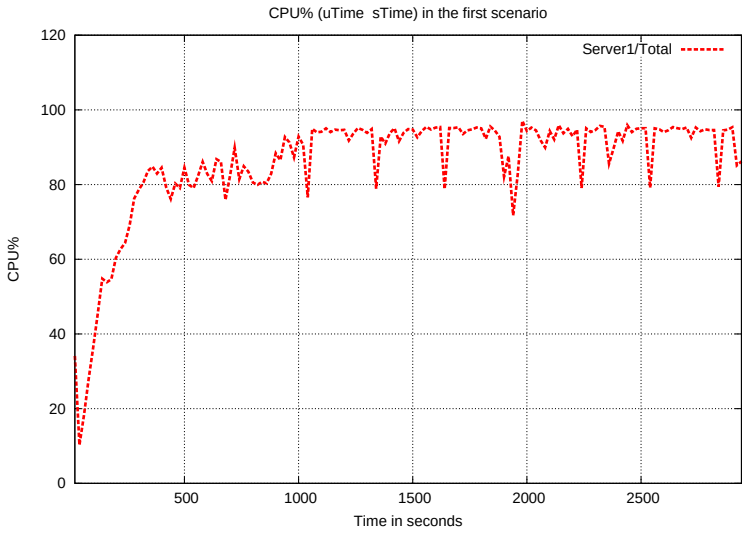



Figure 4.9: Total CPU usage by server1.

Figure 4.9 shows the CPU utilization of server1. As we can see, it takes the game server some time before it stabilizes. The mean value of server1 is 85.98%[10]. Also, the CPU% drops now and then. The downward spikes in the graph happens every 300 second (checked the log file). They are not that visible in the beginning, but are easier to spot after the server has reached twenty-three players. It is not unlikely that this is caused by Munin. As mentioned, Munin runs every five minutes (which corresponds to 300 seconds) in order to gather data through a cron job.

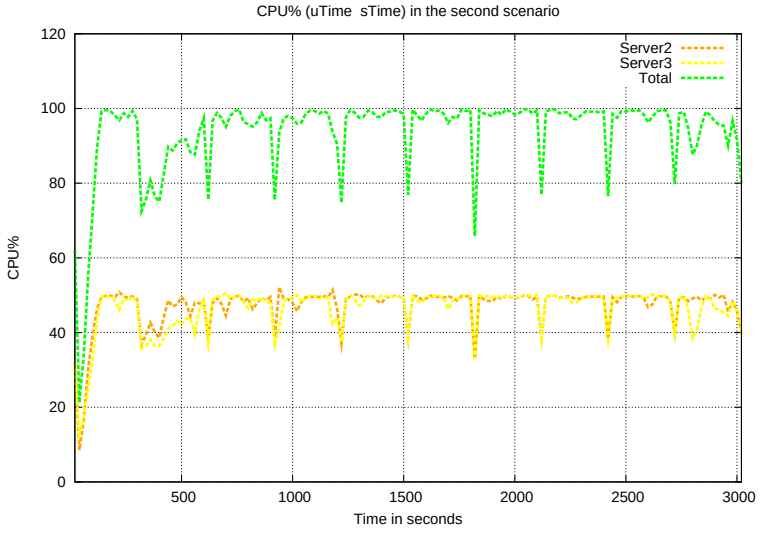



Figure 4.10: CPU usage by server2, server3 and total usage.

[10]Even though the overall performance cannot be summerized into one single number, people tend to do so. This is supported by Lilja[19] who says that the mean values can be useful for performing coarse comparison.

Figure 4.10 shows that both server2 and server3 harmonize with each other in terms of utilized CPU. The mean value is 47.25% for server2, 46.42% for server3. This tells us that the CPU load has been approximately reduced by half, so has players. However, the total mean load is 93.66%. This means that running two servers takes up more CPU than one, even though the number of players connected to the machine remains the same.

The downward spikes assumed to be caused by Munin in scenario one, are also present in this scenario. By checking the log files, the author can confirm that the spikes appears with a fixed interval of five minutes in this case as well. According to the questionnaire which the author filled out during the experiment, the time which the spikes appeared matched the time when heavy lag was observed. This applies to the first four spikes.

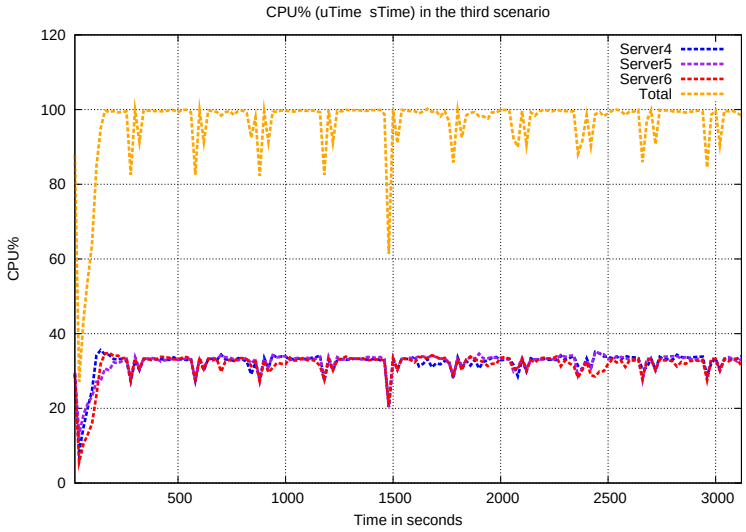



Figure 4.11: CPU usage by server4, server5, server6 and total usage.

Figure 4.11 shows the CPU utilization of server4, 5 and 6. The total mean CPU usage is 96.13%, which tells us that total CPU utilization becomes less efficient each time a server is added. The mean value for server4 is 32.19%, 32.29% for server5, 31.65% for server6. Like scenario two, spikes appear every five minutes here too. However, they do not cause any noticable lag on the server. A possible explanation to this is the fact that Munin does not get that much CPU time compared with scenario two.

### CPU% and Players

We believe that the number of players are affecting the CPU%. To determine the strength of a linear relationship we use correlation coefficient[19]. The formula is:

$$r = \frac{S_{xy}}{\sqrt{S_{xx}S_{y_y}}}. \tag{4.1}$$

where

$$S_{xx} = \sum_{i=1}^{n}(x_i - \overline{x})^2 \tag{4.2}$$

$$S_{y_y} = \sum_{i=1}^{n}(y_i - \overline{y})^2 \tag{4.3}$$

$$S_{xy} = \sum_{i=1}^{n}(x_i - \overline{x})(y_i - \overline{y}) \tag{4.4}$$

The calculated value of r gives us information about the strength and direction of the relationship. The value r gives will always lie between [-1, +1], but we usually differ between three main values[56]. If r = +1, then we a perfect linear correlation between X and Y. The "+" sign tells us that the values of both variables are increasing. When there is no correlation between X and Y we get 0. -1 implies a perfect linear correlation between X and Y. The "-" means that the higher values on one of the variables causes a decrease on the other.

The coefficient of determination ($r^2$) is defined by Lilja[19] as:

*"The fraction of the total variation explained by the regression model."*

$r$ and $r^2$ for each server was calculated in Excel. The results are presented in table 4.2.

| **Server** | **Correlation coefficient (r)** | **Coefficient of determination ($r^2$) $*$ 100)** |
|---|---|---|
| 1 | 0.8889 | 79.01% |
| 2 | 0.7870 | 61.93% |
| 3 | 0.6269 | 39.30% |
| 4 | 0.7447 | 55.45% |
| 5 | 0.6391 | 40.84% |
| 6 | 0.8794 | 77.33% |

Table 4.2: Correlation coefficient and coefficient of determination for each server instance.

As we can see, server2-4 shows a strong relationship. Server1 and 6 shows a very strong positive correlation between CPU% and players. In fact, $r^2$ for server1 tells us that 79.01% and 77.33% for server6 of the variance in CPU% can be explained by their linear relationship. Although there is a very strong correlation in these two cases we cannot conclude that the number of players affects the CPU% for sure as the value varies on each server. A possible explanation to this may be the fact that server1 and 6 has a more smooth increase in players connecting to the server.

The author believes real life data is a more suitable approach to investigate this further. Therefore, a data sample was taken from one of the log files which have been collected throughout the project period. The sample consists of twenty and a half hour data. The result is presented in a scatter diagram[11] in figure 4.12.

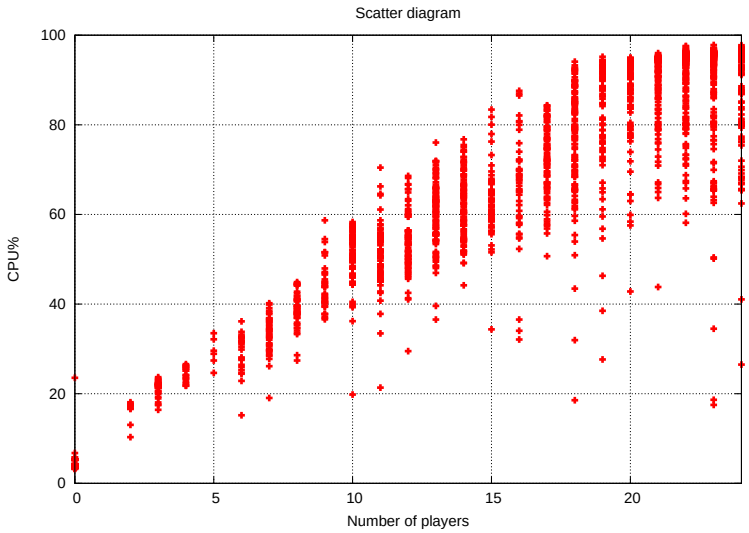



Figure 4.12: Scatter diagram of the real life data sample.

The figure shows that the points are clustered in a positive linear direction. The figure shows the higher the CPU% gets, the more players are on the server. The deviating dots in the figure can be explained in times where all players are ”inactive”, ergo less CPU demanding. From a in-game perspective this could be before a round begins. The players on the BLU team are forced to wait thirty seconds so that RED team can set up defenses.

The correlation coefficient was calculated in Excel in this case as well. The result was r = 0.98083. This tells us that we have a almost perfect linear relationship between the two variables. The coefficient of determination tells us that 96.20% ($(0.98083)^2 = 0.9620 * 100$) of the variance in CPU% can be explained by a linear relationship. The remaining 3.8% of CPU% remains unexplained. We cannot conclude for sure that the number of players causes high CPU%, even though we have found a very strong correlation. Hence the expression, ”correlation does not imply causation”. There may be e.g. a third variable that are affecting the CPU% which causes a correlation. But this is unlikely in our case. Because when the number of players present on the server increases, the CPU must work harder in order to process user input and maintain world state consistency for every player.

Based on the same sample, we can use prediction interval to predict the outcome of a new measurement[57]. The equation for a 95% prediction interval is given by:

$$\overline{x(m)} \pm Z_{\frac{\alpha}{2}} * S(m) \tag{4.5}$$

where x(m) is the average CPU as a function of m. m is the number of players. The calculations are left to Excel.

[11] Scatter diagram are used to illustrate the possible relationship between two variables. Each pair value (x,y) is marked in the diagram with a point. The independent variable is placed on the x axis, while the dependent variable is placed on the y axis. The y variable is always assumed to be one affected.

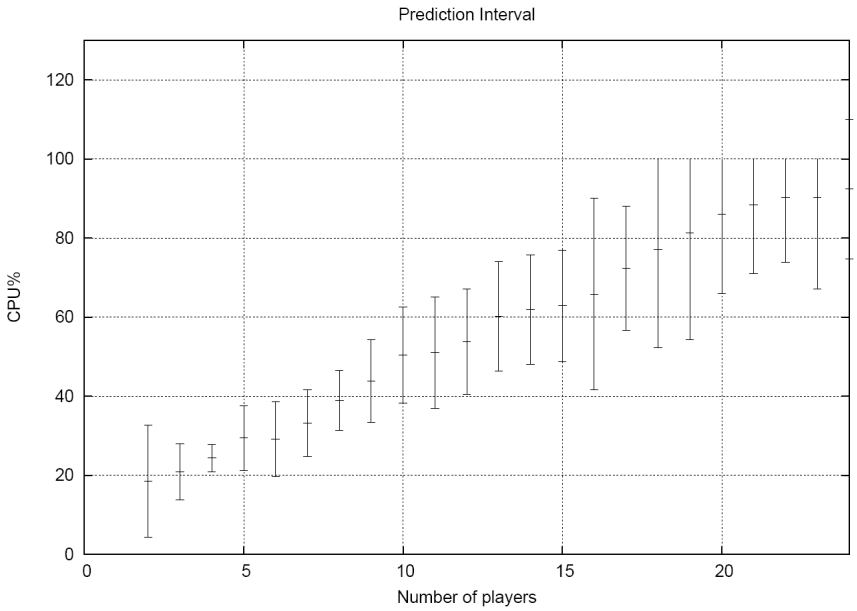



Figure 4.13: Estimation of a new measurement value with a 95% prediction interval (PI).

Approaching 100% of CPU we have to be carefull with a normal distribution since we got a skewed distribution. Upper intervals is limited to 100%. But still the lower interval is set to:

$$\overline{x(m)} - Z_{\frac{\alpha}{2}} * S(m) \tag{4.6}$$

The figure 4.13 shows a linear increase in the predicted values. If we carry out a new measurement with N players, we can say with 95% probability that the CPU value will lie between the upper and lower limit[12]. This allows us to a certain degree predict the CPU load on a single core Intel processor with the same clock rate based on how many players the instance allows.

### 4.2.3 Memory%

The ratio between the processs Resident Set Size (RSS) and the physical memory is higher on server 1 than the rest. The highest achieved memory% utilization by a server was 10.6% (server1). Which means that the process had been allocated circa 161 Megabytes in the main memory by the OS. As we can see in figure 4.14, the servers becomes stable after a short while and remains approximately stable throughout the period.

[12]The regression line (y) was calculated in Excel for each limit. The reason why $r^2$ for mean differs now is because the values for zero players are removed in this case. Upper limit: $y = 4.304x + 16.13$, $R^2 = 0.973$. Mean: $y = 3.597x + 10.70$, $R^2 = 0.992$. Lower limit: $y = 2.889x + 5.278$, $R^2 = 0.958$.

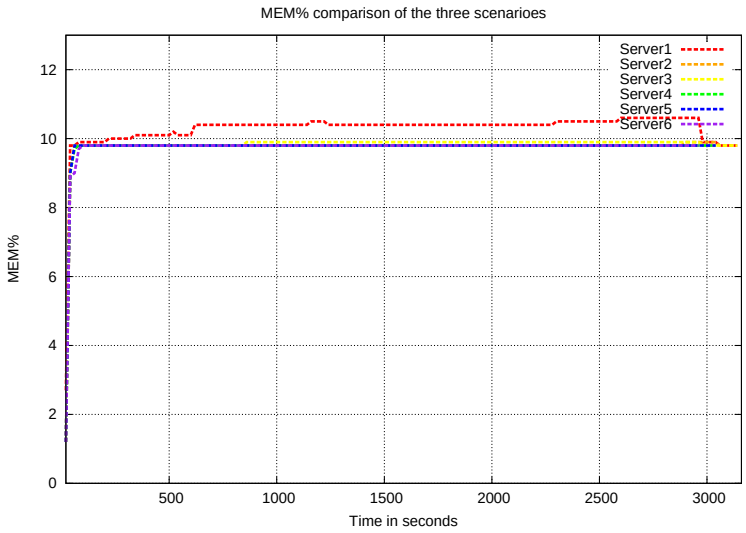



Figure 4.14: Comparing the MEM% in each scenario.

What is interesting is the fact that server1 runs with 24 players, server 2 and 3 with 12 and server 4,5 and 6 with 8. Yet, they use almost the same amount of MEM%. It seems like the game server is allocating the needed memory during its initialisation phase.

### Memory and +map

The command that starts a game server comes with several options:

```
./srcds_run -console -game tf +ip 128.39.74.31 -port 27015 +map cp_dustbowl +maxplayers 24 +exec server.cfg -secure
```

Port, maxplayers, secure and server.cfg are optional, but the rest are required. If map is not used, the server will do nothing (idle). Server 1-6 were executed with the option ”+map dustbowl” and ”+maxplayers 24” (appendix F.1.1) although twenty-four slots was not necessary in the two last scenarios. To examine whether the map option has something to do with the memory usage on the game server, additional measurements were carried out. The monitoring tool were run twenty times to collect data on an empty server, where each test took thirty measurements. The results revealed almost identical values in each test.

| **Measurement Value #** | **MEM%** |
|---|---|
| 1 | 2.7% appeared as first value in twenty-eight cases. The remaining were 2.5% and 2.2% |
| 2 - 30 | 8.9% appeared in all tests. |

Table 4.3: Results from the twenty tests.

This tells us that memory utilization is predictable when there are no players on the server. Also, it seems like the process has a minimum memory requirement, which barely increases as the number of players goes up (figure 4.14).

### Memory and map type

To check whether one specific map uses more memory than another, further measurements lasting ten minutes each were carried out. One measurement for each official map. The results are presented in figure 4.15.

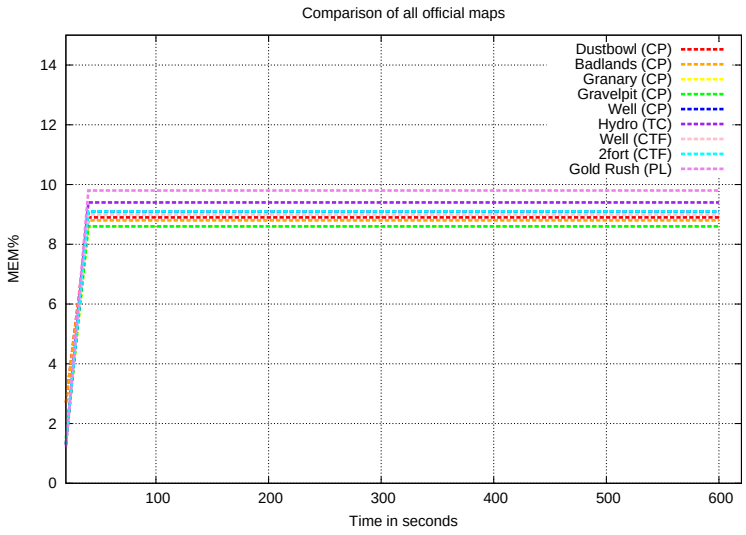



Figure 4.15: Comparing all official maps impact on MEM%.

According to figure 4.15 some map needs more memory than others. The map named ”Gold Rush” is the largest map and requires 9.8% of the installed memory. This is equivalent to ca 149 Megabytes. It is likely that the memory allocated at initialization depends on the map.

## 4.2.4 Resident Set Size

A process consist of several pieces, also known as pages or segments. When a new process starts, the OS takes just a piece(s), typically start instructions of the given process into the real memory (RAM), which gives room for more processes. The part of the process which is at all time in main memory is known as Resident Set. The size (RSS) depends on how much RAM the OS has allocated for this process.

Figure 4.16 shows that server1 occupies most RSS of all servers instances. It had 161.26 Megabytes allocated in the main memory before the server was shut down. As we can see in our case, the greater the number of players is on a server, the more RSS is allocated. Surprisingly, there is little difference between an instance running twenty-four players and a one with eight players.

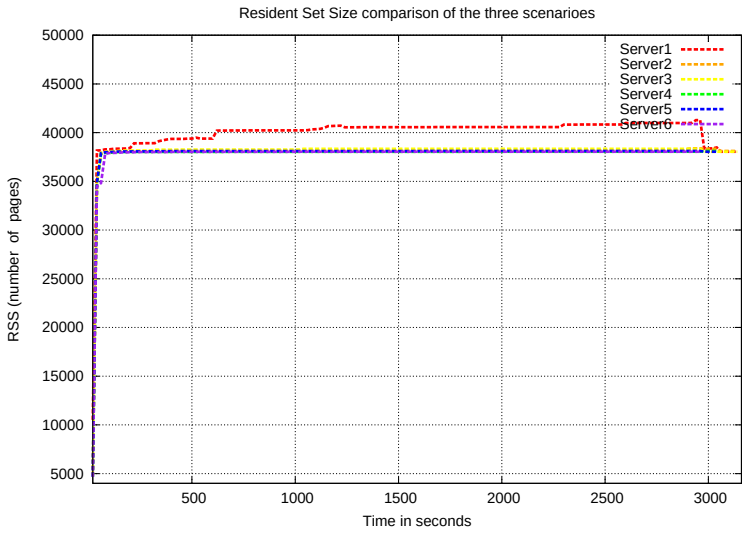



Figure 4.16: Comparing the RSS in each scenario.

### 4.2.5 Minor Faults

Pages which have not been accessed by the process in a while are moved to the OS's free list[58]. To access a page in the free list, the process generates an interrupt (page fault), or in this case a "minor fault". The page is then relinked to the page table before removed from the list. In cases where there are not much available memory left, the pages in the free list are moved to the disk, this is known as swapping.

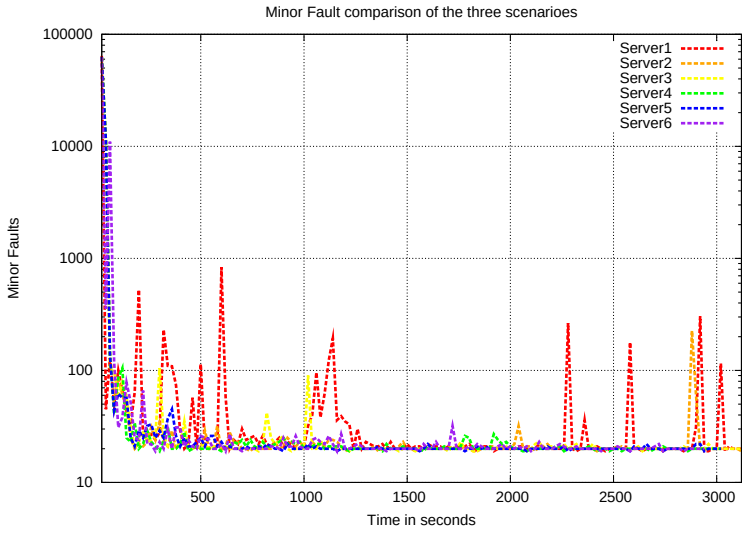



Figure 4.17: Comparing the minor fault in each scenario.

Figure 4.17 shows that the number of minor faults is high in the beginning of each server, before dropping dramatically. This means that the server process allocates all

memory in the beginning. It is irrelevant how many users that are logged in, made so for efficiency. The memory is used actively throughout each scenario. The figure is also an indication that the server has enough memory due to the fact that the number of minor faults are low. Also, spikes do not appear every five minutes, which tells us that Munin is not that aggressive in terms of memory.

### 4.2.6 Virtual Memory

Some processes might be greater than the available RAM on the computer. To overcome a scenario where the physical RAM on the computer is filled up, the OS utilize Virtual Memory. Virtual memory, is as opposed to RAM, a ”unlimited” resource where a flexible or fixed amount of space on the hard disk is treated by the OS as if it were real memory.

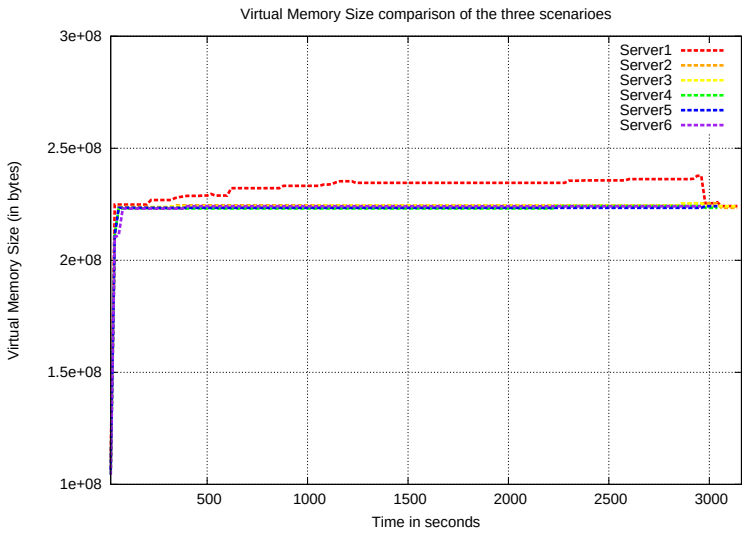



Figure 4.18: Comparing the VSZ in each scenario.

Figure 4.18 shows that server1 uses most virtual memory. Server2 - 6. The maximum virtual memory size achieved on the game server with twenty-four players was 226.68 Megabytes. As we can see, the virtual memory utilization is less efficient as the number of players increases on the server.

# Chapter 5

# Discussion

In this chapter the important findings of the result chapter and its impact are discussed. The five questions raised in the introduction are reviewed. The reliability and validity of the measuring are discussed, in addition to its repeatability.

## 5.1 Server Capacity Planning

The design of the experiment was based on a belief that the number of game servers running would have a great impact on the overall performance. The total mean CPU% value in the first scenario was 85.98%. The answers from the questionnaire tells us that the players had a excellent (9.08/10) game experience at this point. In scenario two, the total mean value increased to 93.66% and a drop in game experience (7.54/10) was reported. In fact, one player would have left the server if it was up to him to decide. Not supricingly, this was an experienced player. In the last scenario, the total mean CPU usage increased by 2.47%, but the overall game experienced was better (8.04/10) in this scenario compared to the previous. Even though the game experience had been improved, the number of people that would leave if they could had increased by one. This was not unexpected as players are an extremely difficult group to satisfy. Surprisingly, by comparing the total CPU% in each scenario we see that adding one extra server instance only contributes to a small portion of the overall CPU load. In that sense, it is obvious that the number of instances are not dominating the load in this experiment. Running both two and three game servers simultaneously with the given player distribution worked well in our case. It did not hamper the game experience to a degree where it becomed unplayable. According to the results from the questionnaire, the major portion of the players were pleased with the service.

So what is dominating the CPU% load? In subsection 4.2.2 we saw that the utilized CPU% on each server decreased as the number of players were reduced by each scenario. It is likely to assume that the players present on the game server has something to do with this. The correlation coefficient between these two variables were calculated for each scenario in order to look into the possible relationship. Generally, people carry out experiments to check whether a correlation exist or not, but in this case this was not captured well. The correlation coefficient varied from one server to another. There

may be several reasons why this happend: (1) multiple server instances are run simultaneously and (2) the script may not have collected data frequently enough to capture the gradually increase in CPU% utilization as players joined the server. Ergo, causing a less noticeable distribution.

The most representative correlation coefficient value was from server1 (table 4.2) where we appear to have a very strong positive (r = 0.8889) linear relationship. It is the most believable value as there is only one server running in this scenario. To check whether this relationship is true for other cases as well, we look into a sample taken from the real life data. The result reveals an almost perfect correlation (r = 0.98083) coefficient value. Ergo, in our case there exist a linear relationship between the number of concurrent players and CPU utilization. This is supported by the findings in Abdelkhalek et al.[7] and Cheng and Ye[59] as well. Additionally, we can make an approximate estimate (figure 4.13) of the load based on how many players that will play on a server.

The game server as an service is not aggressive in terms of memory usage. Server1 (figure 4.14), a instance with twenty-four players had circa 161 Megabytes allocated in the main memory before it was shut down. Based on the fact that there is 1536 Megabytes of RAM installed, the machine should theoretically be capable of hosting at least six server instances (leeway taken into account) at the same time according to these numbers. Most of the memory a process use is allocated in the beginning. This is used actively throughout the period since minor faults are kept at a minimum in each process. By carrying out additional measurements, we find that a empty server is predictable in terms of memory and the amount of memory allocated in the beginning depends on the map. Some maps requires more memory than others. The map which requires the most is ”Gold Rush” (149 Megabytes). A surprisingly results is that a server running twenty four players barely exceeds the memory usage of an empty server. Compared to CPU% utilization, the amount of players on the server hardly affects the memory usage. It is the type of map which dominates the memory usage of a game server rather than players.

## 5.2 Impact

Conventional services like email are different from game server, because there will always exist a demand for these kind of services. A good game server depends on other factors than hardware (maintenance). Getting players to connect to a game server is not an easy task. During the project the author has experienced that the popularity (demand) of a game server is quite fluctuating. It can go from a state where the server is full every day over a period to suddenly remain empty for several weeks. Even though the listed actions in subsection 3.2.1 (Make the Server Attractive) were carried out, this was not enough to establish a longterm popularity. Ergo, a ”community” of players visiting the server on a regularly basis. The author believes true commitment is a keyword in this case, which is hard to achieve when working with the thesis. In that sense, hardware is not that important compared to marketing. It is reasonable to believe that many fail at this point. As a consequence, numerous of servers are left empty.

The fluctuating demand of game servers as a service may complicate things for game server providers. What if each server instance on a machine running a twenty four player limit suddenly becomes full? As we know, gamers as customers are extremely difficult to satisfy since they lose interest if their expectations are not met. Therefore, it is crucial for game server providers in order to reach business goals to provide a top notch services which satisfy their customers expectations. To do so they must plan what kind of hardware they should use in order to prevent CPU or memory bottlenecks, thus degrading the performance and causing lag. Measurements carried out in this thesis have shown that most of the memory allocated to a process happens right after the process has been executed. Also, the amount of allocated memory depends on the map. This allows us to make an worst case scenario estimation on how much memory a machine needs in order to run e.g. four instances on the same physical hardware, thus avoid memory bottleneck. When it comes to CPU, we have shown that there is an almost perfect linear relationship between the CPU utilization and the number of players on the server. This allows us to predict (figure 4.13) the load based on the number of players. This tells us that although the popularity of a game server can be fluctuating, the service is predictable in terms of resources.

## 5.3 Review of The Questions

Five questions were raised in the introduction. To answer these, the author carried out a controlled experiment at school.

### 5.3.1 How many game servers can run simultaneously on one machine?

We have shown that running three game servers with the given player distribution works fine. The number of servers are not dominating the load in this experiment. Adding a game server only contributes to a small increase in the overall CPU load. The question is not how many servers that one can run on one single machine, but how many players that can be present. This decides the number of servers instances the machine can handle.

### 5.3.2 What is the bottleneck that stops us from running one more server?

The experiment shows that players play an important role. By allowing too many players on each running instance, the server will experience a CPU bottleneck. Memory is also a bottleneck, but far easier to predict and therefore avoid.

### 5.3.3 How predictable is a game session in form of resource use?

The results from the experiment did not provide this answer as a map change never occurred. The reason for this was because neither of the teams reached the maximum number of four rounds and the time limit per map in the server configuration file was

set to sixty minutes. However, this can be done in a subsequent experiment using the tools developed in this work.

### 5.3.4 What characterize a server which has no resources left?

The CPU in the experiment was close to being fully utilized. In scenario 2, we saw that the total CPU utilization was close to 100%. The mean value was 93.66%, which tells us that we were close to run out of CPU cycles. Compared to the first scenario, the participants reported a drop in in-game experience. At this point one person would have left the server if he could. In the third scenario, the total CPU reached 100% a couple of times and the mean value was 96.13%. In this scenario two people would have left if they could. It is likely that we have reached the limit for what this machine CPU is capable of handling.

To see what it takes to fully utilize the server's resources, the experiment would have to be repeated. This would have involved inviting the former participants for a new game evening as well as making the server available for all to join (set the game server as public). Additionally, look into the consequence of exceeding the total available memory. This can be done by e.g. run a memory demanding application, execute ten server instances at the same time or switch the RAM chip with a smaller one. Then we would see what characterizes a server which has no resources left.

One thing is for sure though, if either CPU or the memory becomes a bottleneck the server will become unplayable and the players will experience heavy lag. Or in worst case a ”freeze”, were we can assume that the CPU has run out of CPU cycles. When it comes to memory we will see a lot of swapping activity. Swapping out memory to a disk is slow, compared to reside in memory. The players will surely notice lag if they are still there.

### 5.3.5 The best time of the day doing maintenance?

Most of the collected data shows that player activity begins around 5:00 PM and ends after midnight. So, maintenance of the game server should be done before this time.

## 5.4 Reliability and Validity

Both terms are important when it comes to assuring the quality of our measurements. Reliability is whether the same instrument (measuring tool) under the same conditions gives the same result when repeated over time[56]. Thus, the consistency of the results. These results does not need to be identical as the true value may be affected by random error now and then, which again causes a variation in the observed value. There are different ways to test the reliability, but test-retest and internal consistency is the most common. The test-retest technique requires that the measurements should be repeated at least one time. In 4.2.3 (result chapter) the author repeated the measurements of an empty game server twenty times. The figure **??** shows almost identical results. This tells us that the measuring instrument used in this thesis is reliable.

Validity is whether we measure what we want or believe to measure[56]. We measure the performance of a game server with the developed monitoring tool. However, the performance of the game server may be affected by the fact that the tool requires resources in terms of CPU. This is a systematic error. Personal errors also affect the performance. Munin was installed on the machine as an independent check. The author did not expect that running this conventional monitoring tool would affect the performance to that degree it did. Unfortunately, this was not discovered before the experiment was carried out. If time was not an issue, the experiment would have been repeated. Despite of this defect, most results obtained in this study are confirmed by others.

## 5.5 Repeatability

Every component (miscellaneous installs, configuration, scripts++) which have made it possible to carry out the experiment has been carefully documented in the appendix. It is important to do so, as other at some point may want repeat the experiment. A prerequisite for running the monitoring tool is that the game server must run on a Linux platform[13] and have Perl v5.8.8 and Gnuplot v4.2 pre-installed. Both software and OS can be downloaded free of charge.

Repeating the experiment carried out in this thesis is feasible, but those who plan to do this will face the same problems as the author did. In 3.4.2 (Preparations In Advance) three problems were mentioned: how to get twenty-four participants, clients and accounts. That being said, neither participants, clients nor Steam accounts are not assumed to be an issue in their case as these resources probably are ”available” through their respective department. That being said, it may be difficult for others to repeat the same experiment with a game server running the same hardware used in this study, especially the CPU which is now considered to be outdated.

The monitoring tool developed in this thesis can also be modified with ease to support other popular multiplayer online games. For instance, the game named Counter Strike:Source, Day of Defeat: Source or Half-Life 2: Deathmatch can be fully monitored by changing only the map name in fetchSteamInfo subroutine (in the data collection script) to match the maps of the running game server. In case of other games, some modifications of the script are needed. Mainly fetching the right process line in ps aux. Additionally, retrieving in-game information through A2S_info depends on the given game. The developed monitoring tool can be modified to monitor other services than games as well. Once again, it boils down to fetching the right line.

[13]Note: The tool has been tested on Ubuntu only, but is assumed to work with other Linux distributions as long as ps and /proc is available.

# Chapter 6

# Conclusion and Future Work

The main goal in this study was to develop a monitoring system for a game server in order to learn and understand the characteristics of a game server process in a production environment. An other important goal was to identify variables which could give better information or so called decision support. The system should collect data from the variables at a regular interval, thereafter analyze it.

The developed monitoring tool in this thesis gathers data from three places: ps, /proc and the dedicated game server. Based on these data, graphs are generated and presented through a web interface. The graphs on the web page are updated at a given interval. To address the usefulness of the variables, a controlled experiment divided into three scenarios were arranged. The number of instances increased by one for each scenario while the total number of players in each scenario remained fixed.

The results shows that running three Team Fortress 2 dedicated server instances in our case worked well. The number of instances did not dominate the performance, but only contributes to a small increase in the overall CPU load. The more players on a server, the more CPU is utilized. By looking into this we see that the CPU increases linearly with the number of players present on the server. When it comes to memory, an empty server uses nearly the same amount of memory as the server with twenty-four players in the experiment. Ergo, the number of concurrent players hardly affects this resource. Graphs shows that the process allocates most of the memory in the beginning, and this is used actively. For each instance we add, the more physical memory is consumed. Also, we find that the amount of memory allocated depends on which map is set when the server is executed. Knowing this we can make a rough estimate on how much available memory we need in order to run a number of instances on the same physical hardware. That being said, the objective of this thesis has been achieved, but there are room for improvements.

The author consider the possibility of the developed monitoring tool in this thesis being adopted by game server providers to be minimal. Instead, the script may be useful for people administrating their own game server. Compared to Munin, the script takes up a minimum of system resources and provides performance graphs per instance, instead of an overall performance overview. The tool is a good alternative in cases where one want to monitor a game server without affecting the overall performance of the

machine.

## 6.1 Future Work

### 6.1.1 Munin

If time was not a limiting factor the experiment carried out in this study would be repeated again to check whether Munin is the application which causes in-game lag at regular intervals or not. This can be done in various way:

- Make a note of the time whenever a participant put up his hand, which implies that lag has been experienced.
- Improve the questionnaire, as all participants did not write down the time when they experienced lag, just that they had experienced some lag.
- Start Munin simultaneously with the script so that we know the exact time Munin starts.

### 6.1.2 Other Games and Hardware

Another future project would be to monitor the performance of other games than Team Fortress 2 to check whether the findings in this study are representative. Additionally, test with different hardware. It would be interesting to see how a game server performs using different technologies, like multi-core CPUs.

### 6.1.3 Improve Monitoring System

Although the monitoring system works in its current shape, the author have identified two areas for further improvements:

**User friendliness** : The tool consist of three individual scripts where each of the them must interact together in order to make up a fully functional monitoring tool. Additionally, each of the scripts takes several arguments. This relation between the components may confuse an outsider. To ease the execution of the scripts before the experiment, a additional script was made. Merging these scripts with enhanced user friendliness and additional functions can be a project for future work.

**Functionality** : At the time being, the tool plots gathered data without processing it first, ergo it plots raw data. This concerns variables gives cumulative data. For instance variables like stime, utime and minor fault. Consequently, this makes it hard in some cases to interpret the true meaning of what the graphs shows through the monitoring web page. As a result, the cumulative data has to be processed manually which is not a sufficient solution in real life production

environment. Therefore, the tool should automate this process of transforming cumulative data before being plotted.

# Bibliography


[1] Entertainment S. Association. Essential facts: About the computer and video game industry (2006). http://www.theesa.com/archives/files/EssentialFacts2006.pdf, 2006.

[2] Entertainment S. Association. Essential facts: About the computer and video game industry (2007). http://www.theesa.com/archives/files/ESA-EF\%202007.pdf, 2007.

[3] Logitech. Logitech is gaming. http://www.logitech.com/index.cfm/gaming/\&cl=us,en, 2008.

[4] Robert W. Crandall and J. Gregory Sidak. Video games: Serious business for americas economy. http://www.theesa.com/archives/files/2006\%20WHITE\%20PAPER\%20FINAL.pdf, 2006.

[5] eSport Arena. Faq. http://www.esportarena.net/faq, 2008.

[6] Chris Chambers, Wu chang Feng, Sambit Sahu, and Debanjan Saha. Measurement-based characterization of a collection of on-line games. In *IMC '05: Proceedings of the 5th ACM SIGCOMM conference on Internet measurement*, pages 1–14, New York, NY, USA, 2005. ACM.

[7] Ahmed Abdelkhalek, Angelos Bilas, and Andreas Moshovos. Behavior and performance of interactive multi-player game servers. *Cluster Computing*, 6(4):355–366, October 2003.

[8] Microsoft. Overview of performance monitoring. http://www.microsoft.com/technet/prodtechnol/Windows2000Pro/reskit/part6/proch27.mspx?mfr=true, 2008.

[9] Tobias Oetzel. Realtime player statistics for half life 1 and half life 2. http://www.hlstatsx.com/, 2008.

[10] Timo Stripf. Game server browser and administration tool. www.hlsw.org, 2008.

[11] Linpro. Munin. http://munin.projects.linpro.no/, 2008.

[12] Ethan Galstad. Nagios. http://www.nagios.org/, 2008.

[13] Cacti. Cacti. http://www.cacti.net/, 2008.

[14] Valve. Server queries. http://developer.valvesoftware.com/wiki/Server_Queries, 2008.

[15] Aeleen Frisch. *Essential System Administration*. O'Reilly & Associates, Inc., Sebastopol, CA, USA, third edition, 2002.

[16] FAQs. What is black box/white box testing? http://www.faqs.org/faqs/software-eng/testing-faq/section-13.html, 2007.

[17] Testing Brain. Software testing. http://www.testingbrain.com/, 2008.

[18] Lorenz Breu. Online-games: Traffic analysis of popular game servers (counter strike:source). Master's thesis, Eidgenssische Technische Hochschule Zrich, September 2007.

[19] David J. Lilja. *Measuring computer performance: a practitioner's guide*. Cambridge University Press, New York, NY, USA, 2000.

[20] Brookhaven National Laboratory. The first video game. http://www.bnl.gov/bnlweb/history/higinbotham.asp, 2008.

[21] Oilzine.com. Origins of the two pillars of the home gaming market. http://www.oilzine.com/features/features_details.asp?ID=49, 2008.

[22] PDP-1 Restoration Project. Spacewar! http://www.computerhistory.org/pdp-1/index.php?f=theme&s=4&ss=3, 2008.

[23] Discovery Channel. I, videogame. http://www.discoverychannel.co.uk/ivideogame/, 2007.

[24] William Stewart. Unix history. http://www.livinginternet.com/i/iw_unix_dev.htm, 2008.

[25] D. M. Ritchie. The evolution of the UNIX time-sharing system. *BSTJ*, 63, 8:1577–1594, 1984.

[26] Michael Miller. A history of home video game consoles. http://www.informit.com/articles/article.aspx?p=378141, 2005.

[27] Wikipedia. Pong. http://en.wikipedia.org/wiki/Pong, 2008.

[28] Wikipedia. Nintendo entertainment system. http://en.wikipedia.org/wiki/Nintendo_Entertainment_System, 2008.

[29] Hickling Arthurs Low (HAL) Corporation. Entertainment software: The industry in canada. http://www.theesa.ca/esa-whitepaper.pdf, 2007.

[30] Vg Chartz. The most comprehensive videogame charts in the world. http://www.vgchartz.com/, 2008.

[31] Dan Hewitt. U.s. video game industrys growth outpaces national economy. http://www.theesa.com/archives/2007/11/us_video_game_i.php, 2007.

[32] Stephen E. Siwek. Video games in the 21st century: Economic contributions of the us entertainment software industry. http://www.theesa.com/files/VideoGames-Final.pdf, 2007.

[33] Wikipedia. Gross domestic product. http://en.wikipedia.org/wiki/Gross_domestic_product, 2008.

[34] BOXX. We know vfx and it shows. http://www.boxxtech.com/, 2008.

[35] Graham McKenna. Gamers ripe for high-end a/v systems, research shows. http://www.cepro.com/article/gamers_ripe_for_high_end_audio_and_video_systems_research_shows/D3/, 2007.

[36] Blizzard Entertainment. World of warcraft reaches new milestone: 10 million subscribers. http://www.blizzard.com/press/080122.shtml, 2008.

[37] Martyn Williams. Toshiba shows prototype tv running on ps3 chip. http://www.pcworld.com/article/id,141282/article.html, 2008.

[38] Embedded Star. Mercury computer debuts cell be processor for industrial, medical, military. http://www.embeddedstar.com/press/content/2005/10/embedded18990.html, 2005.

[39] Wikipedia. Team fortress 2. http://en.wikipedia.org/wiki/Team_Fortress_2, 2008.

[40] Game Monitor. Team fortress 2 :: Game server / player search. http://www.game-monitor.com/search.php?game=tf2, 2008.

[41] B. Knutsson, H. Lu, W. Xu, and B. Hopkins. Peer-to-peer support for massively multiplayer games. citeseer.ist.psu.edu/knutsson04peertopeer.html, 2004.

[42] Microsoft. Client/server topology. http://msdn2.microsoft.com/en-us/library/bb153244(VS.85).aspx?pull=/msdnmag/issues/0500/security/default.aspx, 2008.

[43] Valve. Source multiplayer networking. http://developer.valvesoftware.com/wiki/Source_Multiplayer_Networking, 2007.

[44] Wikipedia. Game server. http://en.wikipedia.org/wiki/Game_server, 2008.

[45] Jaecheol Kim, Jaeyoung Choi, Dukhyun Chang, Taekyoung Kwon, Yanghee Choi, and Eungsu Yuk. Traffic characteristics of a massively multi-player online role playing game. In *NetGames '05: Proceedings of 4th ACM SIGCOMM*

*workshop on Network and system support for games*, pages 1–8, New York, NY, USA, 2005. ACM.

[46] J. Wu-chang Feng; Chang, F.; Wu-chi Feng; Walpole. A traffic characterization of popular on-line games. *Networking, IEEE/ACM Transactions on*, 13(3):488–500, June 2005.

[47] Olav Dalland. *Metode og oppgaveskriving for studenter*. Oslo : Universitetsforlag, second edition, 1997.

[48] Knut Halvorsen. *Å forske på samfunnet. En innføring i samfunnsvitenskapelig metode.* Bedriftsøkonomens forlag, Oslo, 1993.

[49] Hans Petter Ulleberg. Forskningsmetode og vitenskapsteori (1). `http://www.sv.ntnu.no/ped/hans.petter.ulleberg/vitenskaph99.htm`, 2002.

[50] William M. K. Trochim. Nonprobability sampling. `http://www.socialresearchmethods.net/kb/sampnon.php`, 2006.

[51] Mark Burgess. *Analytical Network and System Administration. Managing Human-Computer Systems*. J. Wiley & Sons, Chichester, 2004.

[52] Muppet. Guide server.cfg. `http://forums.srcds.com/viewtopic/5264`, 2007.

[53] Steam forum. Insane cpu usage. `http://forums.steampowered.com/forums/showthread.php?t=644751`, 2008.

[54] Ellie Quigley. *PERL by example*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, fourth edition, 2008.

[55] William Stallings. *Operating systems : internals and design principles*. Prentice-Hall, Inc., fourth edition, 2001.

[56] Kristen Ringdal. *Enhet og mangfold : samfunnsvitenskapelig forskning og kvantitativ metode*. Fagbokforlaget, Bergen, 2001.

[57] David S. Moore and George P. McCabe. *Introduction to the practice of statistics*. W.H. Freeman and Company, third edition, 1998.

[58] Rick van der Mieden. Summary: what is the minor faults meaning in mpstat. `http://www.sunmanagers.org/pipermail/summaries/2005-August/006675.html`, 2005.

[59] Long Cheng and Meng Ye. System-performance modeling for massively multiplayer online role-playing games. *IBM SYSTEMS*, 45(1):355–366, January 2006.

[60] Japje. Install: Linux (rev. 2). `http://www.srcds.com/db/engine.php?subaction=showfull&id=1098643920&archive=`, 2004.

[61] Row. Tf 2 server install. `http://forums.srcds.com/viewtopic/5151`, 2007.

# Appendix A

# Setting Up a TF2 Server (linux)

Installing the Team Fortress 2 dedicated server combines the Linux tutorial provided by forum member Japje[60] and and row[61] at Source Dedicated Server (SRCDS) forum. It consist of five steps:

Before we start we have to make sure that we are not root. It's better to do it right away than afterwards as you might forget in the process that you are installing an application as root, thus you start over again (personal experience).

**Step 1: Get HLDSUpdatetool**
Created a dir called srcds_1 in my user homedir. Then downloaded the HLDSUpdatetool from Valve with wget in the dir. Then set correct permissions before hldsupdate.bin can be executed. Press "yes" to agree with "Terms and conditions". This gives us a new file called "steam" which also need permission set.

```
mkdir srcds_1
cd srcds_1
wget http://www.steampowered.com/download/hldsupdatetool.bin
chmod +x hldsupdatetool.bin
./hldsupdatetool.bin ,typed "yes"
chmod +x steam
```

**Step 2: Download files**
To install the Team Fortress 2 dedicated server run the commands below. This might take a while depending on your connection. "tf" indicates that we're installing Team Fortress.

```
./steam -command update -game "tf" -dir .
./steam -command update -game "tf" -dir .
```

**Step 3: Configure server**
The next step is to create files in order to customize the gameplay. These are mapcycle.txt, motd.txt and server.cfg.

```
cd orangebox/
cd tf/
nano motd.txt
nano mapcycle.txt
cd cfg/
nano server.cfg
```

**Step 4: Run server**

The download from step 3 produced a new file called steam_run, which allows us to start the server. The code below starts a TF2 server at 128.39.74.31:27015 with cp_dustbowl as the first map. It also set max players allowed on the server to 24 players. Our customized server configuration file is also executed at run.

```
cd ..
cd ..
./srcds_run -console -game tf +ip 128.39.74.31 -port 27015 +map cp_dustbowl +maxplayers
24 +exec server.cfg -secure
```

This command will start a dedicated Team Fortress 2 server on 128.39.74.31:27015. The initial map is cp_dustbowl with 24 players restriction. The command will also execute server.cfg at run in addition to mapchange. Last but not least, in order to make the server ”cheat proof”, Valve Anti Cheat (VAC) is activated (the ”-secure” option).

**Step 5: Maintain server up to date**

Keeping the server up to date is important for two reasons: (1) retrieve the latest patches and (2) enable clients to connect to the server.

```
./steam -command update -game ”tf” -dir .
```

# Appendix B

# Game Configuration Files

Each time a dedicated game server runs it reads from several files. These are server.cfg, motd.txt, mapcycle.txt and autoexec.cfg. The ”server.cfg” file contain information about the server configuration. It also refer to two other files named banned_user.cfg and banned_ip.cfg. These are used to ban players who violate the server rules. ”motd.txt” displays a welcome message when a user connect to the server and ’mapcycle.txt’ defines the map cycle rotation. The ”autoexec.cfg” file specifies that the server should log game related information and send this to a HLstatsX server.

## B.1 Server Configuration (server.cfg)

```
1
2  // this is your server name as shown in the server list
3  hostname ''HiO | All welcome | No lag | Dustbowl/Badlands | HLstatsX''
4
5  sv_password ''''
6  // your server password. a pair of double quotes means it is not set and
7  // anyone can join
8
9  // start rcon settings
10
11 rcon_password ''******''
12 // your rcon password to log into the dev rcon console or HLSW rcon
13 // console
14 sv_rcon_banpenalty 5
15 // Number of minutes to ban users who fail rcon authentication
16 sv_rcon_maxfailures 10
17 // Max number of times a user can fail rcon authentication before
18 // being banned
19
20 // end rcon settings
21
22 // start cvars for balancing un-even teams
23
24 mp_autoteambalance 1
25 // 0 is off and 1 is on. if 1 then should be used in conjunction with
26 // the following 3 commands
27 mp_autoteambalance_delay 60
28 // Time (in seconds) after the teams become unbalanced to attempt to
29 // switch players
30 mp_autoteambalance_warning_delay 30
31 // Time (in seconds) after the teams become unbalanced to print a
```

```
// balance warning
mp_teams_unbalance_limit 1
// Teams are unbalanced when one team has this many more players than
// the other (0 disables)

// end cvars for balancing uneven teams

// start cvars for round and game times

mp_enableroundwaittime 1
// Enable or disable timers to wait between rounds. 0 is off 1 is on
mp_bonusroundtime 20
// Time after round win until round restarts (in seconds)
mp_restartround 20
// Time the current round will restart (in seconds)
mp_stalemate_timelimit 120
// Time limit (in seconds) of the stalemate round
mp_stalemate_enable 1
// Sudden death enables on draw. 0 enables stalemate
mp_timelimit 60
// game time per map in minutes

// end cvars for round and game times

// start cvars for win conditions

mp_maxrounds 4
// Max number of rounds to play before server changes maps
mp_winlimit 0
// Max number of rounds one team can win before a server changes maps

// end cvars for win conditions

// start client specific cvars

mp_forcecamera 1
// force dead clients to first person mode disabling free look. 0 is off
// 1 is on
mp_allowspectators 1
// enable or disable spectators on the server. 0 is off 1 is on
mp_friendlyfire 0
// 0 is off and clients can do harm to team mates. 1 is on and players can
// kill or injure team mates
mp_footsteps 1
// footsteps on or off. 0 is off and 1 is on
sv_cheats 0
// allow cheats to be used by the client. 0 is off 1 is on
sv_timeout 300
// the amount of time in seconds that a client is booted for no input
sv_maxspeed 320
// the maximun speed a client can move at
sv_consistency 1
// Force clients to pass a consistency check for critical files before
// joining server. 0 is off 1 is on
decalfrequency 10
// the pause in seconds between a decal being sprayed

// end client specific cvars

// start cvars for communication

sv_voiceenable 0
// allow players to use a microphone. 0 is off 1 is on
sv_alltalk 0
// toggles whether both teams can hear each others voice comms or not.
// 0 is off 1 is on. recommend it being off
```

```
98  mp_chattime 10
99  // players can chat for this amount of time (in seconds) after a game
100 // is over
101
102 // end cvars for communication
103
104 // start download cvars
105
106 sv_allowupload 1
107 // allow custom decals to be uploaded. 0 is off 1 is on
108 sv_allowdownload 1
109 // allow files to be downloaded from the server. 0 is off 1 is on
110 net_maxfilesize 15
111 // Max download file size. Default is 15
112 sv_downloadurl ""
113 //redirect download location
114
115 // end download cvars
116
117 // start bandwidth rates/settings
118
119 sv_minrate 20000
120 sv_maxrate 30000
121 decalfrequency 10
122 sv_maxupdaterate 100
123 sv_minupdaterate 66
124 sv_mincmdrate 66
125 sv_maxcmdrate 100
126
127 // end bandwidth rates/settings
128
129 // start server logging
130
131 //log off
132 // enable or disable server logging. on is on off is off
133 sv_logbans 0
134 // Log server bans in the server logs
135 sv_logecho 0
136 // Echo log information to the console. 0 is off 1 is on
137 sv_logfile 0
138 // Log server information in the log file. 0 is off 1 is on
139 sv_log_onefile 0
140 // log everything in one file
141
142 // end server logging
143
144 // start cvars for general operation
145
146 sv_lan 0
147 // is this an internet or LAN server. 0 is internet 1 is LAN
148 sv_region 3
149 // server location. -1 is the world, 0 is USA east coast, 1 is USA west
150 // coast, 2 south america, 3 europe, 4 asia, 5 australia,6 middle east,
151 // 7 africa
152 sv_contact StianO.Jelmert@stud.iu.hio.no
153 // Contact email for server admin
154 sv_pausable 0
155 // enables or disables whether the server can be paused. 0 is off 1 is
156 // on
157 sv_pure 1
158 // forces all clients on the server to use content that matches what
159 // is on the server. 0 is off 1 is on
160 sv_pure_kick_clients 1
161 // kicks clients that do not have content that matches what is on the
162 // server
163
```

```
// end cvars for general operation

// start execute ban files

exec banned_user.cfg
exec banned_ip.cfg

// end execute ban files
```

## B.2 Message of The Day (motd.txt)

Before:

> Welcome to Team Fortress 2 @ HiO
>
> Play nice and behave properly;)
>
> Our map rotation is:
> - Dustbowl
> - Badlands

Now:

> http://teamfortress.iu.hio.no/motd.html


Figure B.1: What a user will see after connecting to OUC's Team Fortress 2 server

The screenshot (figure B.1) present information divided into four sections:

**Overview:** The number of unique players the server have had, kill and headshots since 04.02.08. In addition to the current map, map time and the number of players on the server when connecting to the server.

**Top 10:** The ten best players on the server.

**Information:** Brief information about OUC.

**Server rules:** Five server rules that should be followed by those who want to play on the server.

### B.2.1 motd.hml

```
1
2  <html>
3  <head>
4  <title>MOTD</title>
5
6  <style type="text/css">
7  body
8  {
9          background-color: #f4f4f4;
10 }
11 .tekst {
12         font-weight: bold;
13         font-size: 10px;
14         font-family: Verdana, Arial, Helvetica, sans-serif;
15 }
16 .tekst2 {
17         font-size: 10px;
18         font-family: Verdana, Arial, Helvetica, sans-serif;
19 }
20 .strek
21 {
22         border-bottom: #000000 1px solid;
23 }
24 a
25 {
26         color: #000000;
27         font-size: 10px;
28         font-family: Verdana, Arial, Helvetica, sans-serif;
29         text-decoration: none;
30 }
31
32 a:link, a:visited
33 {
34         color: #000000;
35         font-size: 10px;
36         font-family: Verdana, Arial, Helvetica, sans-serif;
37         text-decoration: none;
38 }
39
40 a:hover
41 {
42         color: #000000;
43         font-size: 10px;
44         font-family: Verdana, Arial, Helvetica, sans-serif;
45         text-decoration: underline;
46 }
47
48 </style>
```

```
49
50 </head>
51
52 <body>
53
54 <table width="800" border="0" align="center" cellpadding="0" cellspacing="0">
55 <tr>
56 <td align="center"> <span class="tekst">Welcome to</span></td>
57 </tr>
58 <tr>
59 <td align="center">
60 <iframe frameborder="0" src="http://hio.hlstatsx.com/status.php?width=400
61 &server_id=1&game=tf&show_players=0&show_logo=no&map_image=0&show_top=10
62 &bg_color=f4f4f4&border_color=f4f4f4&body_color=f4f4f4&show_summary=1&
63 show_map_wins=0" scrolling="no" width="400px" height="300px"> </iframe>
64 <br><a href="http://hio.hlstatsx.com">For complete stats go to
65 http://hio.hlstatsx.com</a>
66
67 <br><br>
68
69 </td>
70 </tr>
71 <tr>
72 <td>
73 <table width="400" border="0" align="center" cellpadding="0" cellspacing="0">
74 <tr>
75 <td valign="top">
76
77 <DIV valign="top" class=strek>
78 <div align="center" class="tekst">Information</div>
79 </DIV>
80 <div align="center"><span class="tekst2"><br>
81   The server is provided by:<br>
82   <br>
83   <a href="http://www.hio.no/"><img src="hio.gif" alt="HiO" width="392"
84   height="109" border="0"></a><br>
85   </span><br>
86   <span class="tekst2">as a part of a master thesis project at the
87   <a href="http://www.hio.no/content/view/full/21594">Network and
88   System Administration Programme</a>.</span><br>
89   <br>
90 </div></td>
91 <tr>
92 <td colspan="2" valign="top">
93
94 <DIV valign="top" class=strek>
95 <div align="center">
96   <DIV valign="top" class=strek>
97     <div align="center"> <span class="tekst"> Rules: </span></div>
98   </DIV>
99 </div>
100 </DIV><span class="tekst2">
101 <br>
102 1. The administrators are always right.<br>
103 2. Treat other players as you wish to be treaten. <br>
104 2. Player harrassment or offensive behavior will not be tollerated.<br>
105 3. No spam in chat or through voice commands (e.g. need dispenser here). <br>
106 4. Use English or Norwegian language (Swedish and Danish are accepted as well)
107 only <br>
108 5. Play fair and have fun! </span></td>
109 </tr>
110 </table></td>
111 </tr>
112 </table>
113 </body>
114 </html>
```

## B.3 Map Cycle (mapcycle.txt)

```
cp_dustbowl
cp_badlands
```

## B.4 Autoexec (autoexec.cfg)

```
log 1
logaddress_delall
logaddress_add logs10.hlstatsx.com:30653
```

# Appendix C

# Miscellaneous Installs

## C.1 Install New Kernel

```
apt-get update
apt-cache search — grep 2.6
apt-cache search image
apt-cache search linux-image
jed /etc/apt/sources.list
apt-get update
apt-cache search linux-image
apt-get install linux-image-2.6.24-5-server
jed /boot/grub/menu.lst
reboot
```

## C.2 Set Up Web Server

```
sudo apt-get install apache2
sudo /etc/init.d/apache2 restart
http://teamfortress.iu.hio.no/ It worked!
```

## C.3 Enable public_html

```
cd /etc/apache2/mods-enabled
sudo ln -s ../mods-available/userdir.load
sudo ln -s ../mods-available/userdir.conf
/etc/init.d/apache2 restart

http://teamfortress.iu.hio.no/ stianj/
```

# Appendix D

# Emails

## D.1 Academic licensing at Valve

**From: Stian Opsahl Jelmert <StianO.Jelmert@stud.iu.hio.no>**
**To: academiclicensing@valvesoftware.com**
**Subject: Master thesis**

Hi!

I'm a student currently taking a international Masters degree in Network and system administration at Oslo University College, Norway. My thesis is about performance monitoring of a game server, in this case a Team Fortress 2. I?ve now come to the point where i should plan a experiment. I'm thinking of running multiple measurements, first with 2 players, then 4 etc.

My only concern is getting enough participants from my school to participate in the experiment. The experiment will be announced to all students. My school will provide the pc's, but the problem is that i do not have enough steam accounts (with TF2) to all. As an active Steam user for many years, i know that Valve operates with "Guest Passess". Is there a possibility to get guest passes for one day?

Regards,

Stian Opsahl Jelmert

**From: Arsenio Navarro <arsenio@valvesoftware.com>**
**To: Stian Opsahl Jelmert <StianO.Jelmert@stud.iu.hio.no>**
**Subject: RE: Master thesis**

Hello Stian,

Thank you for your email.

I would be happy to help you with your request.

How many guest passeses would you need? How long would you need the guest passes?

Best Regards,

Arsenio Valve Cybercafe Program

**From: Stian Opsahl Jelmert <StianO.Jelmert@stud.iu.hio.no>**
**To: Arsenio Navarro <arsenio@valvesoftware.com>**
**Subject: RE: Master thesis**

Hi!

Hi and thanks again for your reply!

My supervisor and I have now discussed the approach of the experiment in more details. We came up with 24 guest passes lasting a period of 6 weeks. It's hard to assess how much time the testing will take due to installation times on all machines and checks that everything works perfectly before carrying out the experiment. Further, in case there should be a problem i would like to have leeway for two evenings with student participation.

CC'ing to my two supervisors.

Best Regards,

Stian Opsahl Jelmert

**From: Arsenio Navarro <arsenio@valvesoftware.com>**
**To: Stian Opsahl Jelmert <StianO.Jelmert@stud.iu.hio.no>**
**Subject: RE: Master thesis**

Hello Stian,

I can set this up for your. However, instead of using guest passes I would recommend you use Temporary Steam Tournament Accounts. These are the type of accounts that we provide for academic institutions participating in the Valve Academic Licensing Program or SourceU. These accounts operate as normal Steam Accounts. I can provide account usernames as passwords. What are the date you will need access to these accounts?

Best Regards,

Arsenio

**From: Stian Opsahl Jelmert <StianO.Jelmert@stud.iu.hio.no>**
**To: Arsenio Navarro <arsenio@valvesoftware.com>**
**Subject: RE: Master thesis**

Hi Arsenio!

Thanks for all your help, I greatly appreciate it. The date I?ll need these accounts are on April 8th.

EDIT: Can I get the accounts as soon as possible? I am sorry for any inconvenience caused.

Best Regards,

Stian Opsahl Jelmert

**From: Arsenio Navarro <arsenio@valvesoftware.com>**
**To: Stian Opsahl Jelmert <StianO.Jelmert@stud.iu.hio.no>**
**Subject: RE: Master thesis**

Hello Stian,

I will set this up for you effective April 2, 2008.

What is the last day you will need access to the account?

Best Regards,

Arsenio

**From: Stian Opsahl Jelmert <StianO.Jelmert@stud.iu.hio.no>**
**To: Arsenio Navarro <arsenio@valvesoftware.com>**
**Subject: RE: Master thesis**

Hi and thanks for quick reply:)

May 12th would be nice.

Best Regards,

Stian Opsahl Jelmert

**From: Arsenio Navarro <arsenio@valvesoftware.com>**
**To: Stian Opsahl Jelmert <StianO.Jelmert@stud.iu.hio.no>**
**Subject: RE: Master thesis**

Hello Stian,

I have enabled the sequentially named accounts TU0100200PC1 through TU0100200PC24 (TU0100200PC1, TU0100200PC2, TU0100200PC3 etc.) for your use. The password for each account is ”47647581” without the quotation marks. These accounts are associated with your email address. These accounts will be disabled on May 13, 2008.

We would be intersted in seeing your thesis.

Good luck with your project.

Best Regards,

Arsenio

## D.2 Invitation To Game Evening

**From: Stian Opsahl Jelmert <StianO.Jelmert@stud.iu.hio.no>**
**To: stud-iu-4aa-liste@hio.no, stud-iu-5aa-liste@hio.no, stud-iu-1aa-liste@hio.no,**

**stud-iu-1ab-liste@hio.no, stud-iu-1ac-liste@hio.no, stud-iu-2aa-liste@hio.no, stud-iu-2ab-liste@hio.no, stud-iu-2ac-liste@hio.no, stud-iu-3aa-liste@hio.no, stud-iu-3ab-liste@hio.no, stud-iu-3ac-liste@hio.no, stud-iu-1ia-liste@hio.no, stud-iu-2ia-liste@hio.no, stud-iu-3ia-liste@hio.no, stud-iu-1da-liste@hio.no, stud-iu-1db-liste@hio.no, stud-iu-2da-liste@hio.no, stud-iu-2db-liste@hio.no, stud-iu-3da-liste@hio.no, stud-iu-3db-liste@hio.no**
**Cc: Hårek Haugerud <Harek.Haugerud@iu.hio.no>,**
**Kyrre Begnum <Kyrre.Begnum@iu.hio.no>**
**Subject: Team Fortress 2 Game Evening/Spillekveld!**

Hei!

Jeg er en student som for tiden jobber med min masteroppgave i nettverks- og systemadministrasjon. Fra klokken 17:00 til 21:00 førstkommende mandag (14 april) vil det bli arrangert en spillekveld på skolen som en del av oppgaven min. Dette vil finne sted i 4 etasje, rom PH422 i P35 bygningen. Når dette er sagt trenger jeg 24 frivillige studenter. Skolen stiller med PC til hver enkelt deltaker, men det er også mulig ta med egen laptop/desktop så fremt Team Fortress 2 er installert på forhånd (!). Som takk for at dere hjelper meg vil det bli servert pizza og brus den aktuelle dagen.

Ta kontakt om dette hœres interessant ut:)

PS: Maskinene har ikke musmatte, så jeg anbefaler alle om å ta med en.

Mvh,

Stian

Greetings fellow students!

I'm a student currently working on my Master Thesis in the Network and System Administration. On Monday (April 14), from 5:00 p.m. to 9:00 p.m. there will be held a game evening at school (room PH422, 4th floor in the P35 building) as a part of my experiment. Therefore, i need 24 voluntary students that's up for some gaming. The school will provide PC's, but you may as well bring your own laptop/desktop computer as long it has Team Fortress 2's pre-installed. As a reward for helping me out on my thesis there will be served pizza and soda.

Please contact me if this sounds interesting:)

PS: I recommend everyone to bring a mousepad, as they are not available on the datalab.

Best Regards,

Stian

## D.3 Response To The Requests

Hei!

Takk for at du meldte deg på. Setter stor pris på det. Jeg har nå registrert deg som deltaker på mandag. Hvis du mot formodning ikke kan møte opp påmandag er det viktig at jeg får beskjed da jeg er avhengig av deltakere for få gjennomført eksperimentet mitt.

PS: Det er mulighet for å ta med egen pc/laptop dersom du vil det (øker spillopplevelsen). Bare si fra dersom du bestemmer deg for det, da jeg må gi deg brukernavn og passord på forhånd (hvis du ikke har egen konto). Du kan da spille Team Fortress 2 gratis frem til 13 mai.

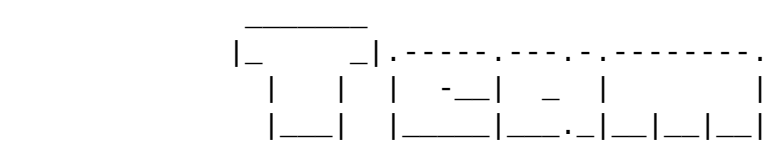
Mvh,

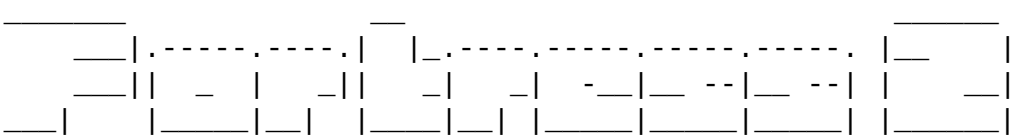
Stian

# Appendix E

# Letters

## E.1 Experiment at School





## Velkommen til spillekveld her på HiO!

Idag er du en del av et eksperiment hvor hovedmålet er å stressteste en Team Fortress 2 spillserver. Eksperimentet tar for seg 3 scenarioer:

- Kjøre 1 spillserver på en maskinen.
- Kjøre 2 spillservere samtidig på en maskin.
- Kjøre 3 spillservere samtidig på en maskin.

For å starte Steam, gjør følgende:

1. Trykk **”Start”**.
2. **”Alle programmer”**.

3. **”Steam”**, og så **”Steam”**.
4. Logg inn med passordet **”47647581”**, deretter trykk **”login”**.

Før vi går inn i Team Fortress må du aktivere **”console”**. Dette gjøres ved å følge stegene nedenfor:

1. Velg fanen **”My Games”**.
2. Høyreklikk på **”Team Fortress 2”**.
3. Velg **”Properties”**.
4. Klikk på **”Set launch options”**.
5. Skriv **”-console”** i vinduet som åpner seg.
6. Klikk **”OK”**, deretter **”Close”**.

For å starte spillet, dobbelklikk på **”Team Fortress 2”** under ”My Games”. Ettersom vi ikke akkurat har med spillmaskiner her å gjøre , anbefaler jeg deg å velge lavere oppløsning i selve spillet for å bedre spillopplevelsen. Dette gjøres ved å følge stegene nedenfor:

1. Klikk på **”options”** i menyen som befinner seg på venstresiden.
2. Velg fanen **”video”**.
3. Sett **”resolution”** til **”800 x 600”**, trykk så **”apply”**.
4. Trykk på knappen **”advanced”**.
5. Sett **”antialiasing mode”** til **”none”**, trykk så **”ok”**.

Hvis du vil justere følsomheten på musen velg fanen **”mouse”**. Anbefaler også å hake av **”mouse filter”**. Trykk så **”apply”** og deretter **”ok”** for å komme tilbake til consolen.

Før vi setter igang med å spille, skal dere deles inn i 3 grupper: A, B og C.

Gruppe A består av PC 1 - 8 (venstre rekke).
Gruppe B består av PC 9 - 16 (midtre rekke).
Gruppe C består av PC 17 - 24 (høyre rekke).

**Scenario 1 starter 17:30 og slutter 18:20:**

Her skal alle gruppene koble seg til serveren ved å skrive følgende i consolen:
connect 128.39.74.31:27015;password hiorocker

**Scenario 2 starter 18:25 og slutter 19:15:**

Her skal gruppe A og halve B (PC 9-12) koble seg til ved å skrive følgende:
connect 128.39.74.31:27015;password hiorocker

Her skal gruppe C og halve B (PC 13-16) koble seg til ved å skrive følgende:
connect 128.39.74.31:27016;password hiorocker

**Scenario 3 starter 19:20 og slutter 20:10:**

Gruppe A kobler seg til serveren ved å skrive følgende:
connect 128.39.74.31:27015;password hiorocker

Gruppe B kobler seg til serveren ved å skrive følgende:
connect 128.39.74.31:27016;password hiorocker

Gruppe C kobler seg til serveren ved å skrive følgende:
connect 128.39.74.31:27017;password hiorocker

## E.2 Questionnaire

# Spørreskjema

1. Hvor erfaren er du med multiplayer FPS spill (TF2, CSS, HLDM)?

_____ Erfaren
_____ Ikke erfaren

2. Hvor ofte spiller du FPS spill?

_____ Hver dag eller nesten hver dag
_____ Flere dager i uken
_____ En dag i uken

_____ 1 til 3 dager per måned
_____ Aldri eller nesten aldri

Scenario 1:

1. Hvordan opplevde du flyten i spillet når du spilte? 10 er best og 0 er dårligst!

_____

2. Hvis du spilte på denne serveren på fritiden og spillingen ikke var en del av eksperimentet, ville du da fortsatt å spille eller ville du ha byttet til en annen server?

_____ Ja, ville fortsatt å spille
_____ Nei, ville byttet

3. Har du endret instillinger (grafikk, tastatur ++) i forhold til det som ble foreslått i arket?
_____ Ja
_____ Nei

Kommentar: Gjerne skriv ned tidspunkt når du opplever lag eller spesielle situasjoner (kortvarige lag).

---

Scenario 2:

1. Hvordan opplevde du flyten i spillet når du spilte? 10 er best og 0 er dårligst!

_____

2. Hvis du spilte på denne serveren på fritiden og spillingen ikke var en del av eksperimentet, ville du da fortsatt å spille eller ville du ha byttet til en annen server?

_____ Ja, ville fortsatt å spille
_____ Nei, ville byttet

3. Har du endret instillinger (grafikk, tastatur ++) i forhold til det som ble foreslått i arket?
_____ Ja
_____ Nei

Kommentar: Gjerne skriv ned tidspunkt når du opplever lag eller spesielle situasjoner (kortvarige lag).

_____

Scenario 3:

1. Hvordan opplevde du flyten i spillet når du spilte? 10 er best og 0 er dårligst!

_____

2. Hvis du spilte på denne serveren på fritiden og spillingen ikke var en del av eksperimentet, ville du da fortsatt å spille eller ville du ha byttet til en annen server?

_____ Ja, ville fortsatt å spille
_____ Nei, ville byttet

3. Har du endret instillinger (grafikk, tastatur ++) i forhold til det som ble foreslått i arket?
_____ Ja
_____ Nei

Kommentar: Gjerne skriv ned tidspunkt når du opplever lag eller spesielle situasjoner (kortvarige lag).

_____

# Appendix F

# Scripts

## F.1 Shell Script

### F.1.1 execute.sh

```
1
2  #!/bin/sh
3
4  ###############################################################################
5  ############ Simple script to start and stop game servers in ############
6  ###################### three different scenarios. ######################
7  ###############################################################################
8
9  # Paths go here.
10 dir="/home/stianj/srcds_1/orangebox"
11 dir2="/home/stianj/"
12
13 # Names of the logfiles.
14 scenario1log1="x1"
15 scenario2log1="x2"
16 scenario2log2="x3"
17 scenario3log1="x4"
18 scenario3log2="x5"
19 scenario3log3="x6"
20
21 # Specifying server variables.
22 com1="+ip 128.39.74.31 -port 27015 +map cp_dustbowl +maxplayers 24 \
23 +exec server.cfg -secure"
24 com2="+ip 128.39.74.31 -port 27016 +map cp_dustbowl +maxplayers 24 \
25 +exec server.cfg -secure"
26 com3="+ip 128.39.74.31 -port 27017 +map cp_dustbowl +maxplayers 24 \
27 +exec server.cfg -secure"
28
29 ###############################################################################
30 ## Clear screen before starting.
31
32 clear
33 echo "------------------------------------------------------------"
34 echo "Start, runs a team fortress 2 server w/ analyzing & logging".
35 echo "Stop, terminates everything which was initialized with Start".
36 echo "Also, the number indicate the number of instances which will \
37 start and stop. E.g. start2 = two game servers are executed."
38 echo "------------------------------------------------------------"
39 echo ""
40 echo "Usage: 'basename $0' {start|stop|start2|stop2|start3|stop3}"
```

```
41
42 read NUM
43
44 case $NUM in
45 "start")
46 sleep 1
47 echo "Starting Team Fortress 2 game server.."
48 cd $dir
49 screen -A -m -d -S server1 ./srcds_run -console -game tf $com1
50
51 sleep 1
52 echo "Initializing Logging.."
53 cd $dir2
54 # runs datacollection script with <IP> <PORT> <LOGFILE>
55 screen -A -m -d -S log1 ./datacollection.pl -I 128.39.74.31 -p 27015 \
56  -L $scenario1log1.log
57
58 sleep 5
59 echo "Initializing Analyzing.."
60 # runs analyze script with <logfile> <interval>
61 screen -A -m -d -S analyze1 ./update.pl $scenario1log1 300
62 ;;
63
64 "stop")
65 echo "Terminating screens.."
66 screen -S server1 -X quit
67 screen -S log1 -X quit
68 screen -S analyze1 -X quit
69 echo "All scripts are now stopped"
70 ;;
71
72 "start2")
73 sleep 1
74 echo "Starting two Team Fortress 2 game servers.."
75 cd $dir
76 screen -A -m -d -S server1 ./srcds_run -console -game tf $com1
77 screen -A -m -d -S server2 ./srcds_run -console -game tf $com2
78
79 sleep 1
80 echo "Initializing Logging.."
81 # runs datacollection script with <IP> <PORT> <LOGFILE>
82 cd $dir2
83 screen -A -m -d -S log1 ./datacollection.pl -I 128.39.74.31 -p 27015 \
84 -L $scenario2log1.log
85 screen -A -m -d -S log2 ./datacollection.pl -I 128.39.74.31 -p 27016 \
86 -L $scenario2log2.log
87
88 sleep 5
89 echo "Initializing Analyzing.."
90 # runs analyze script with <logfile> <interval>
91 screen -A -m -d -S analyze1 ./update.pl $scenario2log1 300
92 screen -A -m -d -S analyze2 ./update.pl $scenario2log2 300
93 ;;
94
95 "stop2")
96 echo "Terminating screens.."
97 screen -S server1 -X quit
98 screen -S server2 -X quit
99 screen -S log1 -X quit
100 screen -S log2 -X quit
101 screen -S analyze1 -X quit
102 screen -S analyze2 -X quit
103 ;;
104
105 "start3")
106 sleep 1
```

```
107 echo "Starting three Team Fortress 2 game servers.."
108 cd $dir
109 screen -A -m -d -S server1 ./srcds_run -console -game tf $com1
110 screen -A -m -d -S server2 ./srcds_run -console -game tf $com2
111 screen -A -m -d -S server3 ./srcds_run -console -game tf $com3
112
113 sleep 1
114 echo "Initializing Logging.."
115 # runs datacollection script with <IP> <PORT> <LOGFILE>
116 cd $dir2
117 screen -A -m -d -S log1 ./datacollection.pl -I 128.39.74.31 -p 27015 \
118 -L $scenario3log1.log
119 screen -A -m -d -S log2 ./datacollection.pl -I 128.39.74.31 -p 27016 \
120 -L $scenario3log2.log
121 screen -A -m -d -S log3 ./datacollection.pl -I 128.39.74.31 -p 27017 \
122 -L $scenario3log3.log
123
124 sleep 5
125 echo "Initializing Analyzing.."
126 # runs analyze script with <logfile> <interval>
127 screen -A -m -d -S analyze1 ./update.pl $scenario3log1 300
128 screen -A -m -d -S analyze2 ./update.pl $scenario3log2 300
129 screen -A -m -d -S analyze3 ./update.pl $scenario3log3 300
130 ;;
131
132 "stop3")
133 echo "Terminating screens.."
134 screen -S server1 -X quit
135 screen -S server2 -X quit
136 screen -S server3 -X quit
137 screen -S log1 -X quit
138 screen -S log2 -X quit
139 screen -S log3 -X quit
140 screen -S analyze1 -X quit
141 screen -S analyze2 -X quit
142 screen -S analyze3 -X quit
143 ;;
144
145 *)
146 echo "Invalid command is entered!"
147 echo "Usage: 'basename $0' {start|stop|start2|stop2|start3|stop3}"
148 ;;
149 esac
150                                                                    #
151 ##########################################################################
```

## F.2 Perl Script

### F.2.1 datacollection.pl

```
1
2  #!/usr/bin/perl
3
4  ###############################################################################
5  ############ Simple script to log data from a game server #############
6  ###############################################################################
7
8  use IO::Socket;
9  use Getopt::Std;
10
11 # Define options.
12 my $opt_string = 'P:L:I:S:p:';
13 getopts( "$opt_string", \my %opt ) or die "Usage: [ -P <PID> ] -I <IP>
14 [ -L <LOGFILE> ] [ -S <SCREEN-SESSION>  ] [ -p <PORT-NUMBER> ] \n";
15
16 # Get the name of the screen session.
17 my $screen = $ARGV[0];
18
19 # port number is set to static unless supplied by the user:
20 my $port_number = $opt{p};
21 $port_number = 27015 unless $port_number;
22
23 #
24 if ( not $opt{I} ){
25     die "You must supply an IP address\n";
26 }
27
28 # Get hour and date in epoch format.
29 $timeanddate = time;
30
31 # Define paths.
32 my $TF2path = "/home/stianj/srcds_1/orangebox/";
33 my $data_path = "/home/stianj/data";
34
35 if ( not $opt{P} and $opt{S} ){
36
37 # we need to know where the binary is stored and where to put the
38 # log-files the name of the data log-file is "$session.log" if
39 # that file exists, we refuse to run:
40
41   if (not $opt{L} and stat("$data_path/$screen.log")){
42                 die "Datafile $data_path/$screen.log already exists\n";
43   }
44
45 # Jumping to the correct directory.
46
47     chdir("$TF2path");
48     print "Starting server $screen on port $port_number\n";
49
50 # Start TF2 dedicated server.
51     system("/usr/bin/screen -A -m -d -S $screen ./srcds_run -console
52     -game tf +ip $opt{I} -port $port_number  +map cp_dustbowl
53     +maxplayers 24 +exec server.cfg -secure");
54
55 # We wait one second because the process might not have started yet.
56      sleep 1;
57 }
58
59 # Get port number.
60 my $variabel;
61 if ( $opt{P} ){
```

```
62      $variabel = ‘ps aux | sed -n -e "/[a-z] *$opt{P} /p"‘;
63  } else {
64          # looks like we can supply the port number and fetch the correct
65          # line...
66      $variabel = ‘ps aux | sed -n -e "/\\.\\/srcds_i486 .*
67      -port $port_number/p"‘;
68  }
69
70  chomp($variabel);
71
72  # we run the script as long as the server runs.
73  while ($variabel) {
74                  my $currenttime = time;
75      print "sarting new  iteration\n";
76
77  # Split and stores the elements from the variable
78  # in a array.
79      @array = split(/\s+/,$variabel);
80
81  # PID is used to collect additional data.
82      my $proc_stat = ‘cat /proc/$array[1]/stat‘;
83      chomp($proc_stat);
84      $proc_stat =~ s/\d+ \S+ \S \d+ \d+ \d+ \d+ \d+ (.*)$/$1/g;
85
86  # Save path from command line to a variable.
87      $file = $ARGV[0];
88
89  # Open log file in append mode.
90      print "Opening Log file\n";
91      my $logfile = "$data_path/$screen.log";
92      if ( $opt{L} ){
93          $logfile = "$data_path/$opt{L}";
94      }
95      open(LOG,">>$logfile") or die
96      "The reason $logfile could not be opened is: $!";
97
98  # Write to log file.
99      my $logline = "$currenttime $array[8] $array[1] $array[2] $array[3]
100     $proc_stat " . fetchSteamInfo() . "\n";
101     print LOG $logline;
102     print "writing line: $logline";
103
104 # Close log file.
105     close(LOG);
106
107 # Print date each time.
108     print "Starting sleep at: ";
109     system("date");
110
111     sleep 20;
112     $count++;
113     print "updating \$variabel\n";
114     $variabel = ‘ps aux | sed -n -e "/\\.\\/srcds_i486 .* -port
115     $port_number/p"‘;
116
117     chomp($variabel);
118
119 }
120
121 print "Server is down, we are done\n";
122
123
124 sub fetchSteamInfo {
125
126 # Establish socket to connect to the game server.
127 my $socket = IO::Socket::INET->new(
```

```
                                Proto=>"udp",
                                PeerPort=> $opt{p},
                                PeerAddr=> $opt{I}
                                 )
  or die "Can't make UDP socket: $@";

# Send A2S_info query packet to server:
$socket->send("\xFF\xFF\xFF\xFFTSource Engine Query\x00");

# Retrieve a2s_info packet reply from server and print respons
# Steam uses a packet size of 1400 bytes + IP/UDP headers.
    my $respons;

# Sometimes we will not receive a response packet from the
# server. Instead of waiting endlessly the script waits 4.
# seconds before setting -1 -1 -1 as a value.
    $TIMEOUT = 4;
    eval {
        local $SIG{ALRM} = sub { die "alarm time out" };
        alarm $TIMEOUT;
        $socket->recv($respons, 1400);
        alarm 0;
        1;
    } or $respons = "";

    if ( $respons ){

# If the packet is received before 4 seconds has elapsed,
# then we unpack it.
        ($a,$type,$version,$hostname,$map,$gamedir,$gamedesc,$appid,
        $players,$maxplayers,$bots,$dedicated,$os,$password,$secure,
        $gameversion) = unpack("iACZ*Z*Z*Z*sCCCaaCCZ*",$respons);

# Close socket connection.
        $socket->close();

       if ( $map eq "cp_badlands" ){
           $map_num = 0;
       } elsif ( $map eq "cp_dustbowl" ){
           $map_num = 1;
       }

        return "$map_num $players $maxplayers";
    } else {
        return "-1 -1 -1"
    }

}
                                                                              #
###############################################################################
```

### F.2.2 analyze.pl

```
1
2  #!/usr/bin/perl
3
4  ###############################################################################
5  ############ Simple script to plot data stored in a log file ############
6  ###############################################################################
7
8  # Path to Gnuplot.
9  my $gnuplot = "/usr/bin/gnuplot";
10
11 # Take two path arguments from command line.
12 # EG. data/proc19.log public_html/proc19.
13 my $logfile = $ARGV[0];
14 my $outputdir = $ARGV[1];
15
16 # We have 44 variables.
17 ####################
18 my $COLUMNS = 44;
19 ####################
20
21 # Create hash with title entries.
22 my %TITLE_ARRAY;
23 $TITLE_ARRAY[2] = "Start time";
24 $TITLE_ARRAY[3] = "PID";
25 $TITLE_ARRAY[4] = "CPU (percent)";
26 $TITLE_ARRAY[5] = "Memory (percent)";
27 $TITLE_ARRAY[6] = "Process Flags";
28 $TITLE_ARRAY[7] = "Minor Faults (Process)";
29 $TITLE_ARRAY[8] = "Minor Faults (Process and Children)";
30 $TITLE_ARRAY[9] = "Major Faults (Process)";
31 $TITLE_ARRAY[10] = "Major Faults (Process and Children)";
32 $TITLE_ARRAY[11] = "uTime";
33 $TITLE_ARRAY[12] = "sTime";
34 $TITLE_ARRAY[13] = "cuTime";
35 $TITLE_ARRAY[14] = "csTime";
36 $TITLE_ARRAY[15] = "Priority (nice value plus 15)";
37 $TITLE_ARRAY[16] = "Nice Value";
38 $TITLE_ARRAY[17] = "?";
39 $TITLE_ARRAY[18] = "Time in Jiffies Before Next SIGALRM";
40 $TITLE_ARRAY[19] = "Process Start Time in Jiffies after boot";
41 $TITLE_ARRAY[20] = "Virtual Memory Size (VSIZE)";
42 $TITLE_ARRAY[21] = "Resident Set Size (RSS)";
43 $TITLE_ARRAY[22] = "Current Limit in Bytes on the Process's RSS";
44 $TITLE_ARRAY[23] = "Startcode";
45 $TITLE_ARRAY[24] = "Endcode";
46 $TITLE_ARRAY[25] = "Startstack";
47 $TITLE_ARRAY[26] = "Current Value of esp";
48 $TITLE_ARRAY[27] = "Current EIP";
49 $TITLE_ARRAY[28] = "Bitmap of Pending Signals";
50 $TITLE_ARRAY[29] = "Bitmap of Blocked Signals";
51 $TITLE_ARRAY[30] = "Bitmap of Ignored Signals";
52 $TITLE_ARRAY[31] = "Bitmap of Catched Signals";
53 $TITLE_ARRAY[32] = "Waiting Channel";
54 $TITLE_ARRAY[33] = "Number of Pages Swapped";
55 $TITLE_ARRAY[34] = "Cumulative nswap for child process";
56 $TITLE_ARRAY[35] = "Exit Signal To Be Sent";
57 $TITLE_ARRAY[36] = "CPU number Last Executed On";
58 $TITLE_ARRAY[37] = "?";
59 $TITLE_ARRAY[38] = "?";
60 $TITLE_ARRAY[39] = "?";
61 $TITLE_ARRAY[40] = "?";
62 $TITLE_ARRAY[41] = "?";
63 $TITLE_ARRAY[42] = "TF2 Map";
64 $TITLE_ARRAY[43] = "Players";
```

```
65  $TITLE_ARRAY[44] = "Max Players";
66
67  die "usage: $0 <LOGFILE> <OUTPUT-DIR>\n" unless $ARGV[0] and $ARGV[1];
68
69  # Make a png subdirectory folder.
70  mkdir($ARGV[1]);
71  mkdir("$ARGV[1]/png");
72
73  # Define size of the png.
74  my $x = 640;
75  my $y = 400;
76
77  for ( $i = 2; $i <= $COLUMNS; $i++) {
78
79  # Call Gnuplot program from Perl.
80  # Output lines to Gnuplot until "commands" shows up.
81      open (GNUPLOT, "| $gnuplot") or die "no gnuplot";
82      print GNUPLOT << "commands";
83
84        set term png size $x,$y
85  #     set title "$TITLE_ARRAY[$i]"
86        set output "$ARGV[1]/png/$i.png"
87        set grid
88        set style line 1 lt 2 lc rgb "cyan" lw 1
89        set ylabel "$TITLE_ARRAY[$i]"
90        set xdata time
91        set timefmt "%s"
92  #     set format x "%a"
93        set format x "%H:%M"
94        set xlabel "Time"
95        plot "$logfile" using 1:'$i' title "Colum 1:$i" with lines ls 1
96
97  commands
98
99  # Close Gnuplot program.
100     close(GNUPLOT);
101 }
102
103 # Create index.html file with graphs from logfile.
104 open(FILE,">$ARGV[1]/index.html")
105 or die "The reason test.html could not be opened is: $!";
106
107 print FILE "<HTML>\n";
108 print FILE "<HEAD>\n";
109 print FILE "<TITLE>GRAPHS</TITLE>\n";
110 print FILE "</HEAD>\n";
111 print FILE "<BODY>\n";
112 print FILE "<h2>Performance results from file $logfile</h2>\n";
113 print FILE "<table>\n";
114
115 # Four pictures per row.
116 my $pictures_per_row = 0;
117 my $max_pictures = 4;
118
119 print FILE "<tr>\n";
120 for ( $i = 2; $i <= $COLUMNS; $i++) {
121 print FILE "<td><div align=\"center\">$TITLE_ARRAY[$i]<br>
122 <a href=\"png/$i.png\"> <img border=0 src=\"png/$i.png\"
123 width=\"250\" align=\"middle\"></a></div></td>\n";
124     $pictures_per_row++;
125     if ( $pictures_per_row == $max_pictures ){
126         print FILE "</tr>\n";
127         print FILE "<tr>\n";
128         $pictures_per_row = 0;
129     }
130
```

```
}
print FILE "</tr>\n";
print FILE "</table>\n";

print FILE "</BODY>\n";
print FILE "</HTML>\n";

close(FILE);
                                                                              #
################################################################################
```

### F.2.3 update.pl

```

#!/usr/bin/perl

################################################################################
################## Simple script to update the graphs ##################
################################################################################

# Specify which log file to generate graphs from and how often
# the graphs should be updated.
my $logfile = $ARGV[0];
my $interval = $ARGV[1];

die "usage: $0 <LOGFILE> <INTERVAL>\n" unless $ARGV[0] and $ARGV[1];

while (1) {

        system("./analyze.pl data/$logfile.log
        /home/stianj/public_html/$logfile");

        sleep $interval;
}
                                                                              #
################################################################################
```