ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

TIK Institut für Technische Informatik und Kommunikationsnetze

Lorenz Breu

Online-Games: Traffic Analysis of Popular Game Servers (Counter Strike:Source)

Semester Thesis SA-2007-42
February 2007 to September 2007

Tutor: Dr. Martin May
Supervisor: Prof. B. Plattner

Abstract

As the amount of traffic generated by online games is increasing, models used to simulate network traffic and design or optimize network hardware and protocols as well as implementations of QoS may have to be adapted to consider such traffic. This study examined the network characteristics of three Counter-Strike: Source gameservers running on a single PC in the ETH network yet publicly accessible across the internet, both in terms of technical issues such as packet size as well as in terms of player load and its effects on the traffic footprint. The results agree with previous studies based on other FPS games including Counter-Strike, the predecessor of the game used in this study. It shows that traffic generated by the client is different to that generated by the server, yet both show similar patterns of short packets sent in bursts. Player behaviour plays a major role in shaping game traffic as was to be expected. Several patterns for online game traffic detection are discussed, specifically the use of unique patterns of status request and status information exchanges. In addition some netflow data from the DDosVax project was analysed to confirm that the patterns seen in the trace data are also visible in the netflow data.

Contents

List of Figures

List of Tables

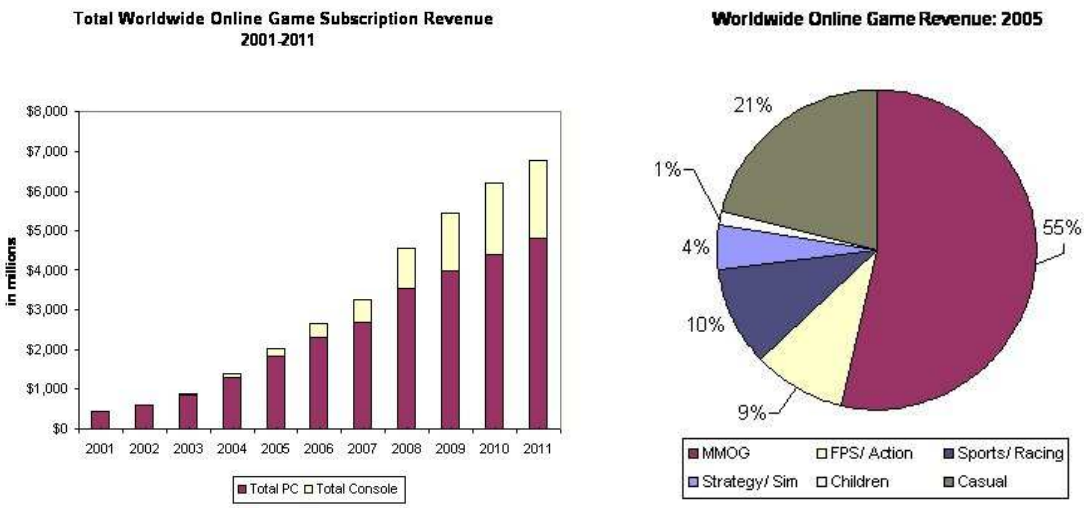
Chapter 1

Introduction

The computer game industry has seen rapid growth since the early 90's due mainly to an increase in processing power of PC's, the increasing capacity of modern graphics cards and the increasingly wide-spread acceptance of computer gaming in society (with two major international leagues and even the existence of professional "gamers" like Jonathan "Fatal1ty" Wendel [1]).
While some numbers suggest (see Figure 1.1) that computer game sales may be stagnating (e.g. [29]) others show that the revenues made from online gaming are ever increasing (see Figure 1.2), especially due to massively multiplayer online games that ask for monthly subscriptions, such as the amazingly popular "World of Warcraft" by Blizzard ([31, 27]). In a press release by PricewaterhouseCoopers released in 2006 the following was said: "Video games and the Internet will be the fastest-growing segments, with compound annual increases of 8.9 and 8.4 percent, respectively. Video games will be propelled by next generation console games and rapid growth in online and wireless games, while increased broadband penetration will enhance Internet access spending and stimulate online advertising." ([30]).

Figure 1.1: Computer sales growth in the U.S., published by the Entertainment Software Association [29]

The data used in Figure 1.2(b) does not include the actual revenues from FPS games as these games are generally sold through retailers and do not require subscriptions to be played online. The figure only includes revenues through online distribution. However as platforms such as Valve's "Steam" ([11]) or EA Games' "EA Link" ([14]) are becoming more widely accepted and various games become available for online purchase and subsequent download these numbers may change in the future. In addition various concepts and attempts to provide advertising within FPS games online are being developped which should also increase the revenues made through FPS games.
In the early days of network gaming players often hosted games such as Duke Nukem 3D, Wolfenstein, Doom 2, Warcraft II and Command&Conquer using a dial-up connection, or simply played in LAN settings. With the increasing availability of higher bandwidth connections and the increasing popularity of multiplayer games, the next generation of games (especially first-person shooters and strategy titles) included multiplayer parts optimized for play over the internet and free dedicated server software, so that a player no longer had to host a temporary session



(a) Revenue forecast

(b) Revenue source

Figure 1.2: Online game subscription revenues, published by DFC Intelligence [31]

but could connect to his favorite permanent server, leading to the success of such titles as Half-Life, QuakeIII Arena, Unreal Tournament and many more. Today broadband connections are widespread and a plethora of companies (e.g. [2, 3]) rent out dedicated game servers to avid players who play an ever increasing collection of multiplayer enabled games including the newest generation of the classics such as Half-Life 2, Doom 3, Warcraft 3, Unreal Tournament 200X and Quake 4.

Even though it has already been shown that using default port numbers to identify the application generating specific network traffic is not a suitable approach ([4]), it is still used ([4, 18]). While server administrators setting up a single game server on an individual machine tend to use the default port for that game, that can no longer be the case when multiple servers run on the same machine or when administrators run the game client on the same machine as the server. With an ever increasing number of games being played online and the fact that ”vintage” games such as StarCraft are still being played by many players and game server providers (e.g. [2, 3]) often host multiple servers (possibly even of different games) on the same computer, it will definitely become more difficult to map a flow originating from a certain port to a distinct game.

With the increase in available bandwidth and the decline in the price of bandwidth, online games will probably account for an increasing percentage of the overall internet traffic, a trend that was first observed in the late 90’s and early 2000’s [4]. Since those measurements were made, the games have been replaced by their successors and new series of successful multiplayer games have emerged. Modern trends such as massively multiplayer online role-playing games (MMORPG’s) and the fact that newest generation consoles such as the Xbox and PlayStation offer online gaming frameworks will probably make the amount of game-related internet traffic increase even more.

Most fast-paced games such as first-person shooters (FPS) require split-second decisions from the player, and a client’s view of the world has to be synchronized with the global view held on the server as fast as possible in order to be able to decide when and if to run, duck shoot and so forth. Thus packet round-trip times (lag) have to be minimized, which means that smaller packets than those produced by e.g. a file-transfer will be expected as the game tries to minimize the amount of data it has to send and in addition applies compression to that data. The effect of lag on a player’s success in an online game as well as his experience of ”fun” have been studied [5] and it has been shown that for games based on the Half-Life engine a lag of 150ms still results in a playable game. Personal experience however shows that a lag above 70ms already hampers skill and that players tend to join servers where there lag is below 50ms if available, leading to geographically localized player bases for servers. Most modern game servers are also set to

drop players with lags above 100ms automatically. While certain methods such as client-side prediction try to make the game balanced for players with a range of lags connected to the same server, the importance of getting packets to and from the server quickly is still vital for acceptable gameplay, which will naturally affect game traffic characteristics and possibly make it detectable amid non-game traffic, as the average packet size within a flow created by game traffic can be expected to be smaller than that of 'background' traffic. In addition the distribution of packet sizes between several flows connected to a single server may show different variation than e.g. flows from different clients connecting to a single FTP server.
Models have been established that can be used to simulate game-traffic in test networks [6, 7, 8], and the overall characteristics of such traffic have been studied [9, 10], mainly for first-person shooters (FPS) with Counter-Strike as the chief representative and real-time strategy (RTS) games such as StarCraft. This project tries to confirm those findings for the newer generation of FPS games. In addition it examines the same characteristics not for a single server but for three individual game servers running on the same physical computer.
The study aims to analyse and describe the characteristics of network traffic generated by three Counter-Strike:Source servers on one physical computer. Player behaviour and a small number of client-side measurements will also be made. The goal is to describe the structure of traffic generated to and from such a server using several methods in order to provide information for future models, quality of service features, hardware design and possibly protocol design. A first look at netflow data directed at the game server (known IP address) tries to determine if the detected patterns can be seen in netflow data.

1.1 Motivation

As an active "gamer" since the 80's, the author has witnessed the rise of the computer game industry and the move from singleplayer games to a point where almost every single newly released game has some kind of multiplayer mode. Indeed many games released today have been developed solely for play on the internet. In addition the author has witnessed the impact of the new massively multiplayer online roleplaying games (MMORPG) on the computer game scene (e.g. World of Warcraft [27]).
The author has also experienced the increase in available bandwidth, personally starting with a 28kps dial-up connection in the early 90's when an hour online cost up to 8.- to a point where a 5Mbps cable connection is available for as little as 50.- per month.
As computing power (multicore cpu's, multicore gpu's, processors dedicated to calculations of game physics...), available bandwidth and computer game popularity increase and the price of bandwidth decreases, it must be assumed that computer games will cause an ever increasing proportion of the total internet traffic. Knowing the characteristics of such traffic and being able to identify game related flows among background traffic could be used by ISP's to increase game-specific QoS for customers that frequently play games or host game servers. In addition, if game traffic becomes more important, it may influence the parameters used to design networking hardware such as routers, as well as make current traffic models used in network simulations more or less obsolete. Taking into account certain behavioural factors of the players such as times of high activity and geographical distribution of players in relation to a specific server may also have an impact on models of game related traffic.

1.2 The Task

The project task can be divided into the following steps:

1.2.1 Administrative and Legal Issues

Hosting a game server on the network of the Swiss Federal Institute of Technology (ETH) must be properly sanctioned by the network administrators. Proper communication with relevant authorities is of the essence. The game servers cannot be deployed without a green light from the responsible parties. Security issues must also be discussed as well as the use of bandwidth for gaming purposes.

As data will be captured over the course of several days with the possibility of turning the project into a more long-term one, the legal implications of capturing the data sent by clients and using it in the subsequent analysis may have some legal ramifications. The project's full legal compliance to Swiss law and ETH policies must be ensured.

1.2.2 Server Setup and Monitoring

As the traffic produced by clients connecting to the project's game servers will be analysed to detect certain patterns, the servers must run smoothly and without interruptions during the period of data capture. The individual game servers must also have enough resources available to grant the clients the best possible gaming experience in order to have a player base that is as large as possible and produces as much data as possible. Factors affecting server performance and player experience must be constantly monitored, server crashes and data loss must be minimized.

1.2.3 Data Capture

Data must be captured at the relevant place with appropriate methods during a well-defined time window. This includes online data capture and representation, both in the light of server monitoring as well as an additional source of data for comparison with later offline analyses. Once the data has been gathered it must be analysed.

1.2.4 Data Analysis

The data gathered during the course of the project must be analysed. Data should then be compared both with the characteristics of general internet traffic as well as with previous findings. In addition, any observed patterns must be discussed for their feasibility as detection criteria.

1.3 Related Work

The interest of computer scientists in game related traffic and its characteristics seemed evident in the late 90's and early 00's when a series of papers on various issues related with game traffic were published, yet little has been published since then. The publication that influenced this project the most was [10], yet other papers dealing with various aspects that are either affected by or themselves affect game related network traffic also provided insights.

1.3.1 Traffic Characteristics and Network Hardware Design

Most papers dealing with the traffic characteristics of online games used Counter-Strike as the object of research. In [10] an extensive set of properties related with a series of games, among them Counter-Strike, was analysed. The researchers looked at a series of properties:

- packet load: client and server mechanisms lead to highly periodic fluctuations in packet load
- packet size and bandwidth: games tend to produce small packets and are designed to saturate the narrowest last-mile link
- client session times: probability density function of player session times seems to fit a Weibull distribution
- player geographic location: a large proportion of players (unexpectedly) connect from across the Atlantic

The research goes on to determine a series of implications of the characteristics of game related network traffic on router design to cope with predictable periodic bursts of small packets.

1.3.2 Models of Game Traffic and Simulators

A series of studies have been done to examine the packet size and interarrival times of packets at game servers as well as on the client side ([8, 6]) in order to provide mathematical models of the network flows generated by clients playing online games on a server. They show that while in general packet sizes and burst interarrival rates of such traffic are predictable due to the nature of the application causing the traffic, individual clients still produce traffic that correlates with the processing power of their computer and the bandwidth at their disposal. [8] procedes to show that the traffic generated during active phases of online gaming can be modelled, yet that the model cannot be used as background noise in the analysis of other flows, only for QoS evaluations. In [16] Färber continues to show how traffic characteristics of client-to-server and server-to-client traffic depend on the number of clients present on a specific server.

1.3.3 Game Traffic Characteristics and Gaming Host Detection

The characteristics of network flows generated by game servers and the clients connecting to them have also been used to propose a new architecture for game-specific Quality of Service (QoS) in [19]. The paper shows that indeed such flows can be detected at a router based on average packet size and interarrival time statistics and suggests the use of this technique to route packets belonging to game traffic with a high priority at the edge of the network where either game clients or servers are located.
A related field of study is the detection of worms based on flow data (e.g. [20, 21, 22]). Using a near-realtime detection framework based on netflow data or other information provided by UDP packets the propagation of an internet worm was monitored. One of these fields may benefit from insights gained in the other (and vice-versa), and adding the detection of game traffic to a near-realtime monitoring framework may benefit certain network operators (e.g. schools, large businesses).

1.4 Overview

This section has given an overview of the area in which the project is situated. Chapter 2 describes the problem in detail, including more information on the game Counter-Strike:Source, the data that was captured and analysed as well as the overall goals. Chapter 3 describes the methods used to gather and analyse the data for this project. It also shows the exact setup of the servers used, both in terms of software and hardware. Chapter 4 details the results obtained using the methods described in chapter 3. Chapter 5 evaluates the findings and compares them to what previous studies in this field have shown. Chapter 6 finally summarizes the project, briefly lists the contributions made and suggests applications of the findings and further work in the area.

1.5 Acknowledgments

The author would like to thank the following people and organizations for their contributions to this project:

- Martin May (the tutor assigned to this project) for providing ideas and technical support
- Dominik Schatzmann for his help with the NetFlow part of this project, especially the use of the Data Mole Framework
- Gamecenter.ch for their help with estimating the required resources for the gameservers and various suggestions concering the configuration thereof
- Thomas Steingruber of the TIK IT team for his quick help with hardware problems
- Juerg Meier for his help with gameserver administration, especially with kicking and banning offensive players

Chapter 2

The Problem

2.1 Game Traffic and Its Importance

Computer games are becoming ever more popular, more complex, and a large proportion of new games released today include at least one multiplayer mode, some are even designed to be purely multiplayer games from the start. While in the early days of multiplayer games most games where hosted in LAN settings or as short-term internet sessions, many first person shooter (FPS) games today provide free server software for both Linux and Windows operating systems which are then run as long-living servers that remain online for days, months or even years for some popular servers. Several webhosting providers also rent out game servers, and there are many specialized game server providers that offer game servers for a wide range of games (e.g. [3]). In addition to the increase in computer games played online game consoles, previously isolated units confined to the living room, are also becoming linked to each other over the internet. Newest generation consoles such as Sony's PlayStation 3 and Microsoft's XboX 360 include a network adapter that allows the user to hook his (or her) console to the respective online platform and compete against other users of the same system across the internet. Clearly such an increase in the popularity of games coupled with the increase in the number of games that are played online as well as the addition of all the newest generation consoles to the game related traffic flowing across the internet is bound to lead to an overall increase in the proportion of network traffic caused by games.
Besides "simple" games such as FPS and strategy titles, where a client connects to a server, plays a game, disconnects and starts a completely new game on reconnect, a new trend has emerged: Massively Multiplayer Online Roleplaying Games (MMORPG). In these games the "world" persists between sessions and continues to evolve and be affected by other players when a client is offline. A wide spectrum of such games is available, from games that simulate "real life" such as "The Sims" to fantasy roleplaying games such as "D&D Online: Stormreach" and especially "World of Warcraft". Such games have become amazingly popular and people spend a lot of time connected to these servers. Virtual goods obtained by players in the game world are even being sold to other players in the real world. Clearly this attracts a different kind of player than the "traditional" shoot-em-up FPS games. This once again leads to an increase in the number of people playing games over the internet and thus of game related network traffic.
While games became more popular and online gaming became widespread, players gathered into teams or "clans" as they are called in the online gaming community, both to compete in leagues and competitions and a variety of smaller "ladders" as well as to play against other clans for fun. In order to coordinate teamplay a range of tools were developped to discuss tactics, find other online teammates and connect to the same server and to communicate. Most new games designed for online play have incorporated communication into the actual game by adding voice-over-ip functionality. Servers also connect to master servers that are used to publish the status and IP addresses of all active game servers and many tools are available to monitor and administer game servers remotely. This of course adds addional traffic to the traffic generated by the clients and game servers updating the global game state. It may also have different characteristics than the "main" game traffic and may thus potentially change the "footprint" of a game's network traffic if the voice-over-ip traffic is produced by the game itself. This aspect however was not considered in detail in this project.

As game related traffic becomes more important in the sense that a larger proportion of the overall internet traffic is generated by game servers and clients it makes sense to incorporate characteristics of such traffic into models and analyses of internet traffic. The development of new hardware, queueing policies, quality of service or even communication protocols for internet traffic may also benefit from taking game traffic properties and the predicted rise in importance of said traffic into consideration already at an early stage of development.
The goal that was to be achieved in this particular project is the characterization of network traffic of a Counter-Strike: Source game server, both in regards to the more technical aspects such as packet size and rate as well as the more social aspects of the players such as origin of the packets (i.e. player location) and session times. In order to get the required data a game server had to be set up within the ETH network that was to attract players. That meant a stable platform with low latencies and packet loss had to be ensured. In addition online monitoring of certain aspects of the game servers as well as the network traffic had to be set in place.
Once the system and all capture and monitoring procedures were running smoothly, data had to be captured for later analysis. In addition, data on ”normal” background internet activity was to be acquired for comparison.
Once the capture period was succesfully completed the acquired data had to be analysed and visualized to detect possible patterns, either in the game traffic itself or in comparing the game server data with that of the rest of the traffic on the internet.

2.2 Administrative and Legal Issues

Besides the aspects of how to capture what kind of data and what to do with it, installation of a game server (let alone three) within the ETH network had to be sanctioned by the system and network administrators responsible for the ETH’s network infrastructure. The server setup had to meet all security standards and not disrupt the day to day network operation, i.e. the server should not create too much load on the network.
Capturing data sent to the server by clients also had to fulfill all legal requirements imposed on the collection and storage of data.

2.3 The Game

2.3.1 Basics

The game analysed in this project is Valve’s Counter-Strike:Source ([13]) based on the Source engine. The game is the official successor to the highly popular (and free) modification of the original Half-Life title. While the original Counter-Strike is still the most popular game on the Steam network with more than 100,000 servers, Counter-Strike: Source is still a popular game and comes in second on the Steam network with more than 40,000 servers ([12]). The game is an FPS that is basically an online-only title, even though the engine provides so-called Bots that can take the place of human players. Players experience the game world through a first-person perspective with a heads-up-display showing them a radar of the level (”map”) with the position of his/her teammates, the currently selected weapon, health and ammunition status and a small crosshair to indicate where the player is aiming (the game calculates spread depending on player spead, position and weapon) as shown in Figure 2.1.
The game is played in two teams, the terrorists and the counter-terrorists, and depending on the level loaded on the server the teams have to complete a task in a certain time frame. The first scenario is "Bomb Defusal" (on maps named de_xyz) where the terrorist team must plant a bomb at a given site or eliminate the entire opposing team and the counter-terrorists have to either eliminate all terrorists before they can plant the bomb or else defuse the bomb before it explodes. The other is "Hostage Rescue" (on cs_xyz maps) where the terrorists must prevent the counter-terrorists from rescueing four hostages by either guarding them until the round time expires or eliminating all counter-terrorists before they achieve their goal, which is to either eliminate all terrorists or guide the hostages to a safe zone. Once a team wins the map is reloaded and a new round starts anew. A player who has been virtually killed enters spectator mode where he (or she) can see the rest of the game through other players’ views. The server loads a new map when the map time expires.

Figure 2.1: Counter-Strike: Source: Begin of a round on de_dust2

Players find a server to join using either an external tool such as the widely used server browser ”HLSW” ([24]) or ”Gamespy Arcade” ([15]) or use the tools available through the Steam platform itself (see Figure 2.2).

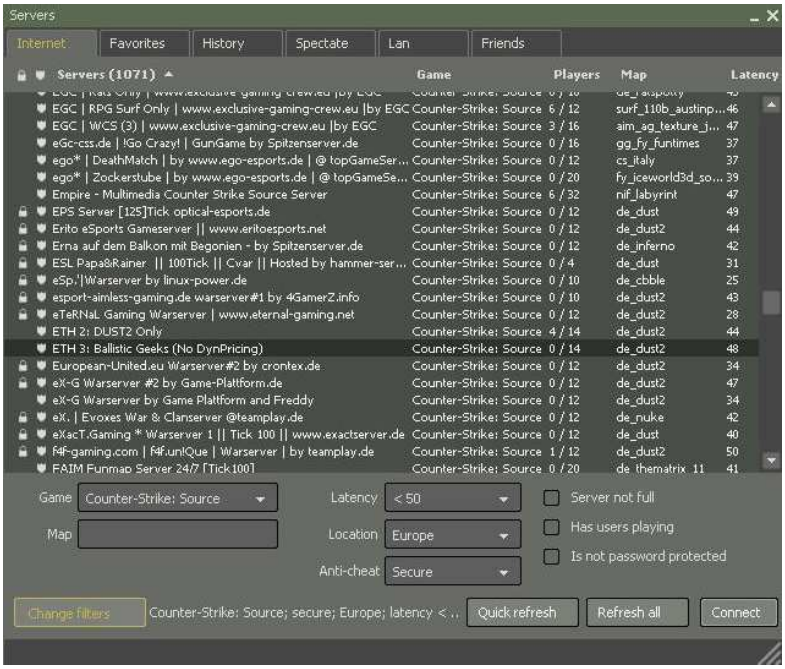


Figure 2.2: Steam server browser interface

2.3.2 Server Software

Counter-Strike: Source uses a client-server architecture, where a central server contains the world state and periodically sends updates of that state to clients, who in turn send events like movement and firing a weapon to the server. The server can either be a dedicated server as it is the case for this project, or a player can host a game while simultaneously playing in it. The

time it takes for a client command to be packed and sent to the server, where it is processed and the time the client receives the game state update from the server is called latency or lag, and most players connect to servers with low lags.
The server is the authority in terms of game rules and all aspects of the game physics, although there are a variety of hacks and exploits trying to give a player certain advantages, such as the ability to see through walls etc. Valve does use its proprietary anti-cheating tool VAC (for Valve Anti Cheat) which it tries to keep up to date faster than new flaws in the game are exploited. Cheating is a major bane in online gaming, not just for electronic sports leagues.
As the game server sends out game state updates to clients at a constant rate ("tickrate") but clients send their packets at varying rates depending on computing power and network bandwidth, several mechanisms are implemented to make game experience a bit more even among clients ([25]). These include client side entity interpolation to smoothen animation and reduce the impact of single packet loss events, client side input prediction, where a movement's effect on player position is applied to the world before the actual "is" state is returned from the server, and lag compensation, where the server estimates the time an event was triggered on the client and moves all entities back to their positions at the corresponding time ([25]).

2.3.3 Patterns and Netflow

Based on the characteristics observed in the traffic specific detection criteria should be discussed. As such detection should be able to happen in (near) real-time, possibly using NetFlow data and a framework like UPFrame ([20]), some of the detected patterns may not be useful for such a purpose, either because calculations and data storage may be limited on the monitoring framework, because the calculations take longer than the interval between two incoming NetFlow records or because the aggregation of packets to flows by NetFlow obfuscate certain packet-level patterns.

Chapter 3

Design

This chapter provides a detailed description of how the problem was solved. All tools and methods used to get and process data are described. Figure 3.1 shows an overview of the data collection and monitoring setup.

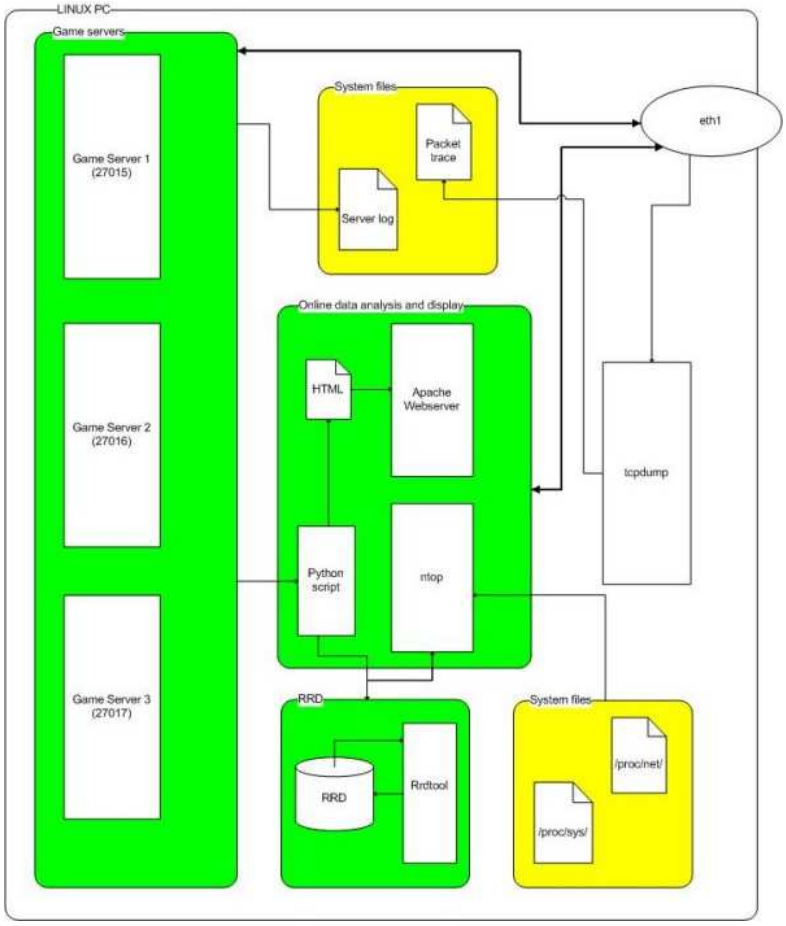


Figure 3.1: Server and data capture setup

3.1 Adminsitrative and Legal Issues

The ”Informatikdienste” of the ETH were informed of the project’s goal and of the fact that a single PC with three game servers was required. The network and system administrators granted the request and allowed the author to setup the servers within the network and for them

to be accessible from the outside by providing a fixed IP. Network traffic was limited to less than 10Mbps/10Mbps by design, as the three game servers would produce less than 7Mbps in either direction, even when under constant maximum load. Administration and monitoring traffic to the server was limited to a minimum.
The legal department also sanctioned the project as no personal information was being collected. While the packet traces may have picked up several bytes of payload for some packets, data produced by a user playing on the game server was not considered to be of personal nature.

3.2 Server Hardware/Software

The server on which the individual game servers ran was kindly provided by the ETH. It was a standard desktop PC with the following specifications:

- Dalco D865 Workstation
- CPU: P4 3.2GHz, 1MB Cache
- 2GB RAM
- 160GB HD

The operating system used was Ubuntu Linux. The computer was appointed the permanent IP of 82.130.102.202.

3.3 Game Servers

In order to determine the aggregate network traffic characteristics of multiple Counter-Strike: Source servers, three dedicated game servers were installed and run on ports 27015, 27016 and 27017 respectively. In addition to the standard game server from the developer the mani-admin-plugin ([23]) was installed for easier administration and enhanced features for connected clients such as voting for map changes. All three servers used an identical mani-admin configuration (see Appendix A.3). The configuration allows standard gameplay with the addition of voting functionality to vote for map changes (for players) or to kick specific players (for admins) as well as facilities to punish players that attack teammates on purpose (again available to admins only). All servers had a maximum player load of 14 and no weapon restrictions were in place. Table 3.1 provides an overview of the main characteristics that distinguish the three game servers.
All servers were run in screens and restarted every morning at 0300h local time to prevent excessive memory leaks as advised by [3]. See Appendix B.1 for details.
Server 1 is a standard CS:S game server with four bots. The bots are replaced by players, and as soon as fewer than four players are on the server bots are added to keep the minimum number of players (as listed by the master server) at four. The seven most popular maps (based on the author's experience and a quick look at the master server list) were in rotation with a map change happening every 40 minutes unless at least 60% of all players vote for a change, in which case the map changes to the map that was voted for at the end of the current round. All standard maps were available for voting. See Appendix A.4 for the configuration file.
Server 2 is a standard CS:S game server with four bots. The bots are replaced by players, and as soon as fewer than four players are on the server bots are added to keep the minimum number of players (as listed by the master server) at four. The server always ran the same map which was deemed the most popular based on the author's experience, no other maps were available, wether in rotation nor for voting. See Appendix A.5 for the configuration file.
Server 3 is a standard CS:S game server without bots. The seven most popular maps (based on the author's experience and a quick look at the master server list) were in rotation with a map change happening every 40 minutes unless at least 60% of all players vote for a change, in which case the map changes to the map that was voted for at the end of the current round. All standard maps were available for voting. See Appendix A.6 for the configuration file.

Administration privileges were granted to the user and one other person in order to tweak server settings according to feedback from players as well as to ban offensive users and players who kept other players from enjoying the game.

	Server 1	Server 2	Server 3
Name:	”DFSOCOM Battlegrounds”	”de_dust2 Only”	”Ballistic Geeks”
Port:	27015	27016	27017
Maps in cycle:	de_dust de_dust2 cs_italy cs_office de_train cs_assault cs_militia	de_dust2 [a]	de_dust de_dust2 cs_italy cs_office de_train cs_assault cs_militia
Maximum players:	14	14	14
Bots:	4	4	0

[a] Only map available on server

Table 3.1: Overview of the game servers

3.4 Online Monitoring

3.4.1 ntop

During the capture phase the program ”ntop” [26] was used to gather data on network activity and keep an eye on traffic. The tool was run with root privileges. It provided detailed information on the relative frequencies of protocols, packet sizes, bandwidth usage etc.. The program also included a plugin for the rrdtool, so the data it collected from the /proc/net system was stored in RRDs and could be displayed online using rrdtool’s graphing function. The program displayed the information using it’s own webserver process using port 3000. ntop was started as a background process as follows:

```
ntop -i eth1 -n -u root &
```

3.4.2 Player Load and Feedback

In addition a small Python script computed cpu load from the /proc/sys system and polled the servers for active players every 10 seconds (see Appendix B.2). The cpu and player load data was stored in RRDs and displayed on a simple homepage served by an apache webserver on the same machine as the game servers. As the script tended to crash without error messages the script was run in a screen and another script was run as a cron job every 5 minutes to check if the monitoring process was still alive, if not it was restarted (see Appendix B.3).
The author also regularly checked if the servers were up and running with the HLSW tool [24] and spent time playing on the servers to check for lag and player feedback.

3.5 Data Collection

3.5.1 tcpdump

The main method for capturing data was a packet trace using tcpdump which was run as a background process during the capture period. The output produced by

```
tcpdump -n -i eth1 -tt -v
```

was written to a text file for analysis after the capture period. The flags ensured that ip addresses were not resolved (-n), timestamps were unformatted (-tt) and additional header data was output (-v).

3.5.2 Delays and Player Load

In addition to the raw packet data collection a script was run in the background to poll the game servers for player load and player info as described in Section 3.4.2. The player load data was written to RRDs. In addition the script retrieved the latency between the game servers and the players as reported by the game servers (”ping”) and sent an icmp ping to each player every 10 seconds, writing the results to a text file for later analysis. If a ping did not ellicit a reply or timed out after 1 second a value of -1 was written (see Appendix B.2).

3.6 Offline Data Analysis

After the packet trace was completed the collected data was visualized using the rrdtool if the data was available in an RRD or else the files created by the tcpdump trace and the python script were parsed using python scripts to produce new files with aggregated data that was then graphed using gnuplot.

3.6.1 Packet Size

The file generated by tcpdump was parsed line by line using a custom python script and the length of each regular IP packet (not ICMP, IGMP or any other such protocol) was added over a certain interval to get the average packet size for that interval. Various plots of time vs. average packet size were generated with gnuplot using varying interval sizes. This was done for all traffic to and from the server and also for each gameserver individually by filtering out packets that do not originate from or are targeted at the gameserver’s specific port. This was done for incoming traffic only, for outgoing traffic only as well as for combined incoming and outgoing traffic.
In addition the distribution of packet sizes was analysed by sorting packets into classes in 10-Byte steps, again using a custom python script. Plots were generated using gnuplot. Distributions were calculated for the aggregate server as well as for each individual gameserver, in incoming, outgoing and combined directions.

3.6.2 Packet Load

Packet Load (Packets/Second) plots were generated for various interval sizes with a custom python script. As with packet size data packet load was calculated for incoming, outgoing and combined data for all three gameservers individually and for all traffic combined.

3.6.3 Pings

The delay (ping) experienced by the client connected to the server was sampled every 10 seconds by polling the gameservers for the value they displayed to the clients inside the game and by sending an icmp ping. Both tasks were done by the same custom python script. Only such clients that did not block the active icmp ping were considered for the analysis. Data was plotted using gnuplot.

3.6.4 Player Load and Session Times

The same script that collected data on delay also collected the current number of players connected to each server and wrote that data into an RRD using the rrdtool module. Data on player load was later extracted using the graphing function of rrdtool. Session times were determined by going through the gameserver log files and collecting all ”connect” and ”disconnect” events. As players which are not satisfied with their performance statistics that are displayed to other

players (kills vs. deaths) they often disconnect and reconnect to reset those statistics. Therefore connects that happened within 5 minutes of the last disconnect by the same client were considered to be extending the current session. Data was extracted from the logs with a python script and the data was then plotted using gnuplot.

3.6.5 Client-Server Flows

In addition to data collected on the server a tcpdump of a thirty minute session during a period of high activity was made using ethereal ([32]) on the PC of the author (Windows XP Professional). At the same time a new packet capture was initiated on the server. Besides the game client the author was also running a tool for friends to find the server he was playing on and a communication program for voice communication with another player on the server. The data collected on the client was reduced to all traffic sent to the server and incoming from it, and on the server the traffic coming from and going to the particular client was extracted from the total capture. The two data sets were then compared using a python script to determine any differences in packet size and load. All plots were made using gnuplot.

3.6.6 Game-External Traffic

Another capture using ethereal was run on the author's machine, this time collecting data on packet size and rate not only from the client but also from the communication tool "Teamspeak" ([33]). Teamspeak uses a client-server architecture, in this case the server (located in Switzerland) rented by the author was used. The data was analysed with the same python scripts used for the other traces and again visualized using gnuplot. During the trace the experimental servers were all empty so the author connected to a random server which was running the map de_dust2, had a maximum player load of 22 players and was almost full during the entire trace lasting from 12:23:45 until 12:49:57 on July 1st, 2007. During the trace only one other player was on the same Teamspeak channel, yet communication between the author and that player was extensive as both players were on the same server and in the same team.

3.6.7 Player Location

A list of unique IP addresses was extracted from the server logs over the packet capture period using a python script. A matching of theses addresses to geographic latitude and longitude coordinates was then done using the GeoIP module for python and the GeoLiteCity database, both provided by Maxmind ([28]). The coordinates were then plotted on top of the world map provided by the standard distribution of gnuplot.

3.6.8 Netflow Data

Once certain patterns were discovered in the traffic based on the packet trace and server log files, NetFlow data provided by the DDosVax Project ([22]) was analysed using a framework developed by Dominik Schatzmann ([34]) for data retrieval and processing of DDosVax data. All traffic captured by the SWITCH network routers was examined for the same time interval (with some additional time before and after to align the data to complete days) as the packet trace and the average packet size of incoming traffic to the game server as well as average packet rates per unique IP for incoming traffic was extracted by filtering the data and only processing records of flows directed at the gameserver's IP address (82.130.102.202). This was done as an initial test to see if any of the patterns and characteristics observed in the packet traces were also visible in NetFlow data.

Chapter 4

Results

4.1 Overview

The packet trace lasted from Thursday May 10 2007 14:33:16 till Wednesday May 16 2007 10:24:18. A total of 90043735 packets were captured (in/out), 86982272 (96.6%) of which were directed at or sent from one of the three ports used by the game servers for game-traffic. Of these the second server was by far the most popular, having 69308908 (79.68%) of the game-related packets either directed at it or produced by it (see 4.1).

	Server 1	Server 2	Server 3	All Servers
In	6950892	32560411	1382898	40894201
Out	7732854	36748497	1606720	46088071
Total	14683746	69308908	2989618	86982272

Table 4.1: Overview of packets to and from gameservers

Of all IP packets captured a vast majority were sized between 64 and 128 Bytes (Figure 4.1). The Figure clearly shows how relatively small packets were sent to and from the server, most being smaller than 256 bytes.

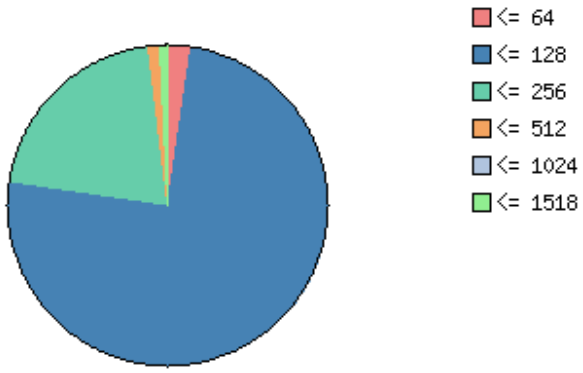
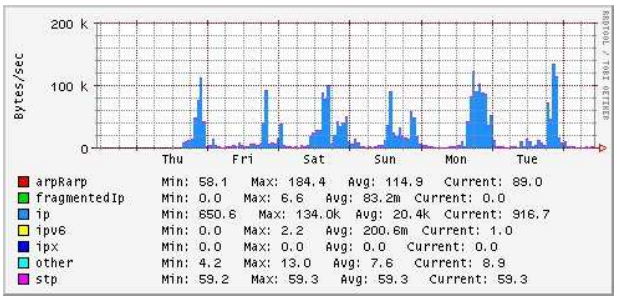
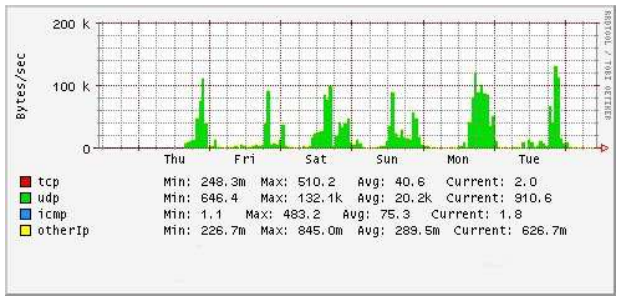


Figure 4.1: Overview: Packet size distribution

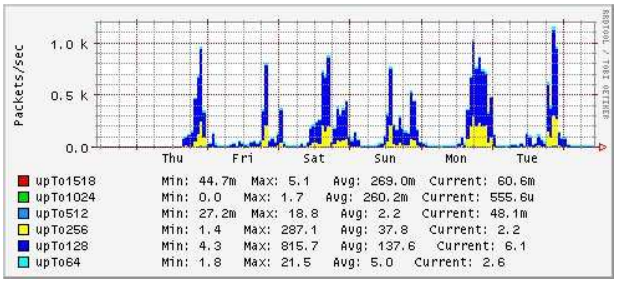
Throughput in Bytes/second showed a clear periodicity with traffic being close to zero in the late night/early morning and rising to a peak every night (Figure 4.2). Figure 4.2(c) shows that the same pattern is apparent in the packet load. Figure 4.2(c) again shows that small packets are the norm. Figures 4.2(a) and 4.2(b) show that UDP was the main protocol used throughout the entire trace.



(a) Throughput (Datalink Layer)



(b) Throughput (Transport Layer)



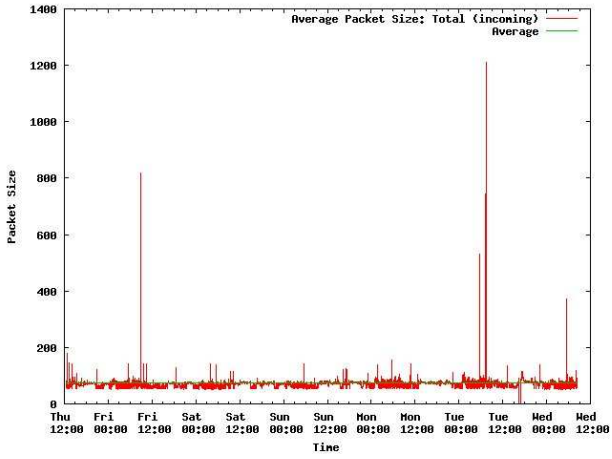
(c) Packet Load and Size

Figure 4.2: Overview: Throughput

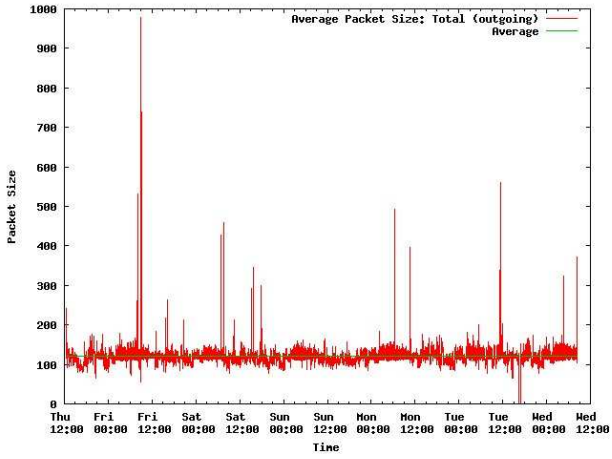
4.2 Network Characteristics

4.2.1 Average Packet Size

There is constant traffic to and from the server when averaging over all packets received within a 60s interval (Figure 4.3). Packet size shows some fluctuation, yet no clear correlation with server activity in terms of connected players is visible at this scale. Packet sizes tend to be relatively close to the average. The average size of incoming packets (75.124 Bytes) is lower than that of outgoing ones (120.328 Bytes).



(a) incoming (75.124 Bytes average)

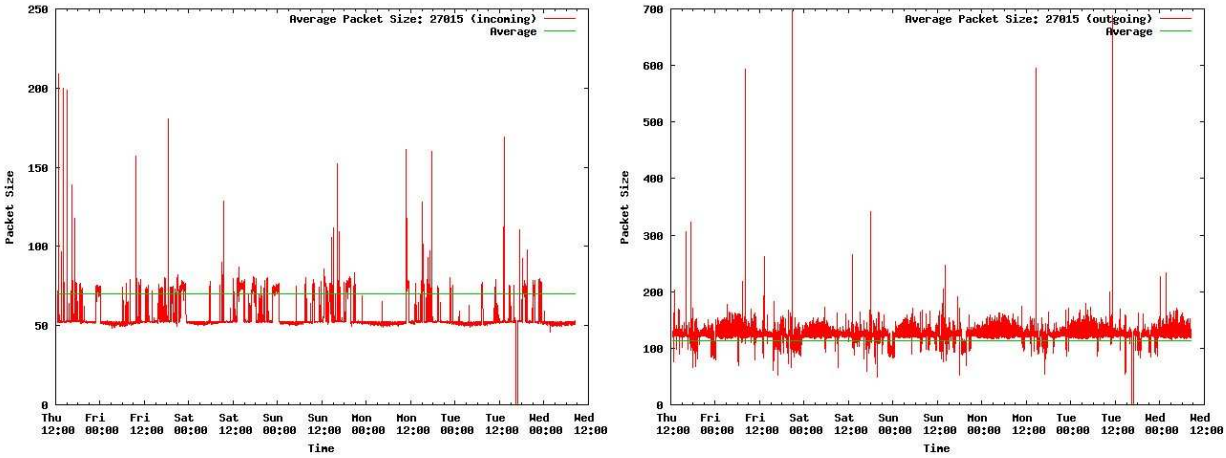


(b) outgoing (120.328 Bytes average)

Figure 4.3: Average packet size for all gameservers calculated over 60s intervals for the complete trace duration

Figure 4.4 shows the average packet sizes over the entire trace for the first server (ETH1) which

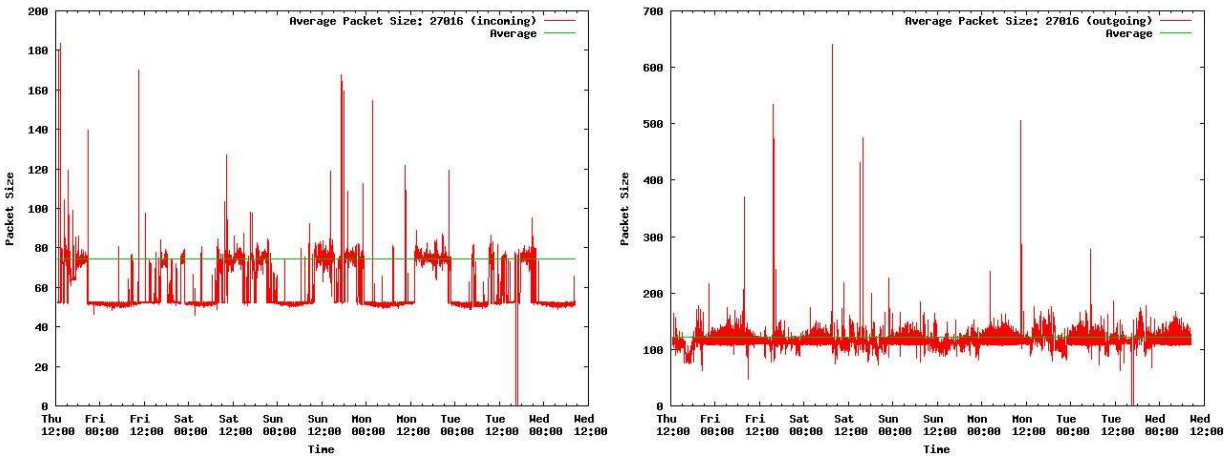
was not very popular. It shows that a constant incoming flow of packets of 53 Bytes is maintained even in times when no clients are connected to the server. This can be explained by the server browser of clients, which sends packets of 53 Byte size to the server requesting server status (number of players, current map, etc.) which in turn sends the requested information back to the server in a 131 Byte packet. Outgoing packets tend to be larger than incoming ones (113.298 Bytes vs 70.144 Bytes).



(a) incoming (70.144 Bytes average) (b) outgoing (113.298 Bytes average)

Figure 4.4: Average packet size for gameserver ETH1 calculated over 60s intervals for the complete trace duration

Figure 4.5 shows the same data for the highly popular server ETH2. The same incoming flow of 52 Byte sized packets during inactive times can be seen. Again the outgoing packets (121.997 Bytes) are larger on average than the incoming ones (74.32 Bytes). The average packet sizes for both incoming and outgoing packets is slightly higher than those of the less popular gameserver ETH1

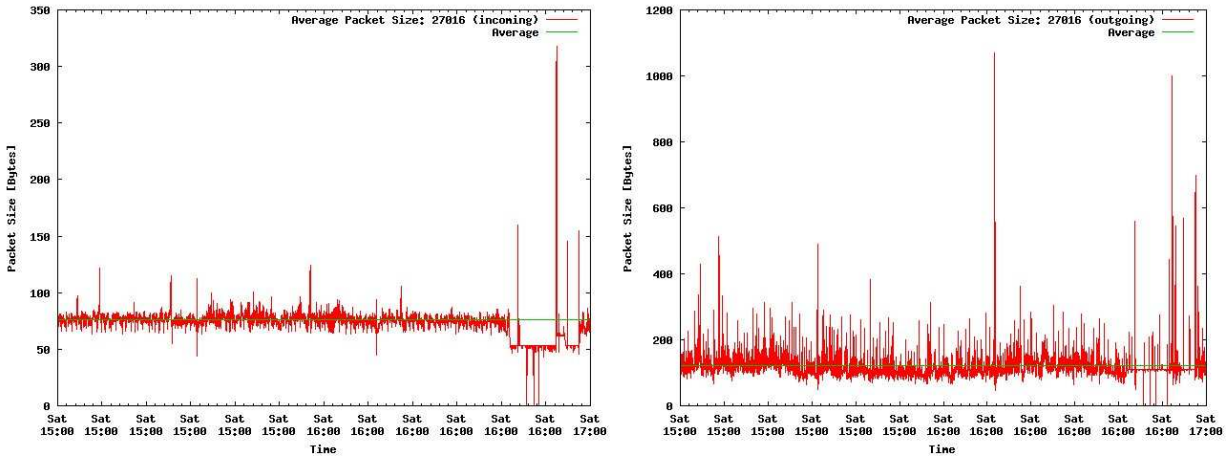


(a) incoming (74.320 Bytes average) (b) outgoing (121.997 Bytes average)

Figure 4.5: Average packet size for gameserver ETH2 calculated over 60s intervals for the complete trace duration

Figure 4.6 shows average packet sizes for the most popular server aggregating packets over 1s for a 2h period. Incoming packets, while smaller on average (75.736 Bytes) than the outgoing

ones (121.125 Bytes), have less variation than the outgoing packets. A dip to 52 Bytes in the average packet size is visible for incoming packets towards the end of the 2h period. Average sizes at this level of resolution are the same as for the lower resolution shown above.



(a) incoming (75.736 Bytes average)

(b) outgoing (121.125 Bytes average)

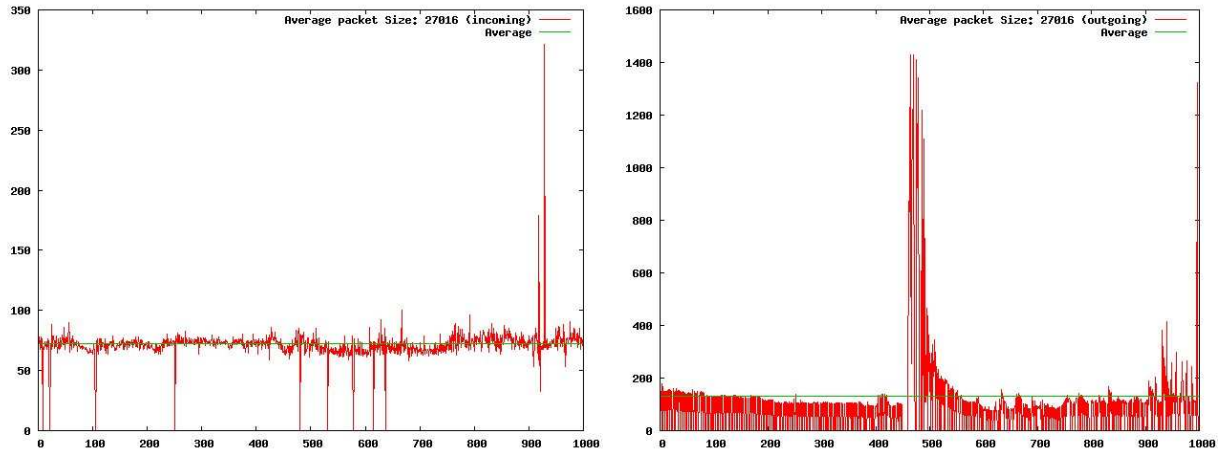
Figure 4.6: Average packet size for gameserver ETH2 calculated over 1s intervals for a 2h period of high activity

Figure 4.7 shows the average packet size calculated over 10ms for a period of 10s during a period of heavy activity. Again incoming packets (72.002 Bytes) are smaller on average than outgoing ones (129.119 Bytes) and these values are in accordance with the ones calculated at lower resolution. The spikes in packet size in the middle of the time interval and again at the very end are probably caused by a ”round start” event and a client connection respectively. The following is the excerpt from the server log for that period:

```
L 05/12/2007 - 14:59:59: World triggered "Round_End"
L 05/12/2007 - 15:00:01: rcon from "82.130.102.202:36261": command "status"
L 05/12/2007 - 15:00:04: "(:oHo:)_::/*ijefij*\::_<178><STEAM_0:0:14263310>
<TERRORIST>" triggered "Got_The_Bomb"
L 05/12/2007 - 15:00:06: World triggered "Round_Start"
L 05/12/2007 - 15:00:09: "Illusive<177><STEAM_0:1:10359392><CT>" disconnected
(reason "Disconnect by user.")
L 05/12/2007 - 15:00:09: "Illusive<190><STEAM_ID_PENDING><>" connected,
address "82.134.47.38:27021"
L 05/12/2007 - 15:00:10: "Illusive<190><STEAM_0:1:10359392><>" STEAM USERID
validated
L 05/12/2007 - 15:00:11: rcon from "82.130.102.202:36265": command "status"
```

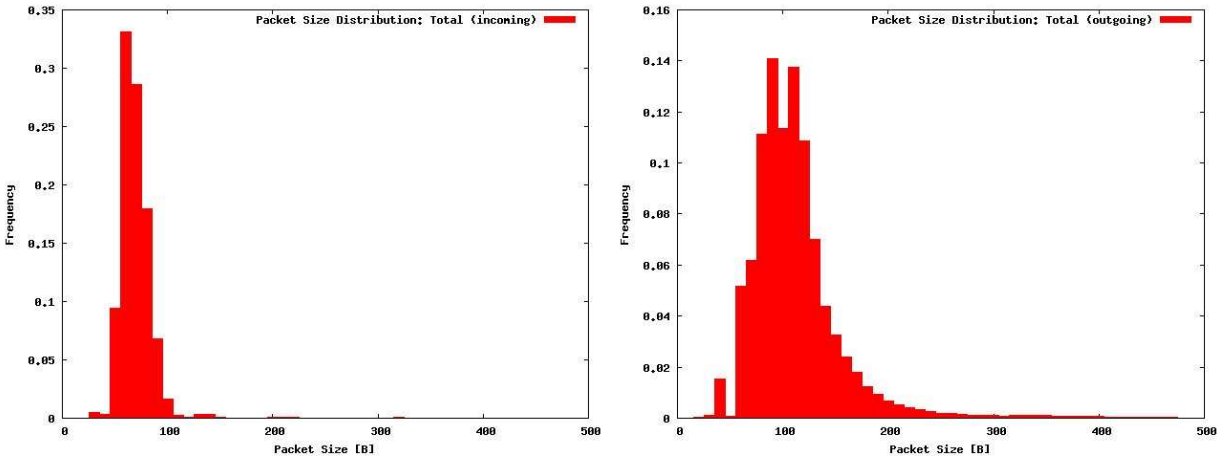
Figure 4.8 shows the packet size distribution of the aggregate game server in both in and out directions. The incoming traffic with a third of all packets between 60 and 70 Bytes has 79,7% of all packets in the range of 60-90 Bytes. The distribution is in general very narrow. The outgoing traffic on the other hand show a long tail of larger packet sizes and a less narrow peak. With a peak at the 90-100 Byte size class 61.2% of packets are between 80-130 Bytes, yet there are still a lot of packets outside this range. The distributions for individual servers show the same characteristics.

When analysing the average incoming packet size by client, it turns out that almost all clients send packets with a length of 50-60 Bytes (Figure 4.9).



(a) incoming (72.002 Bytes average) (b) outgoing (129.119 Bytes average)

Figure 4.7: Average packet size for gameserver ETH2 calculated over 1s intervals for a 2h period of high activity (y-axis: Packet Size, x-axis: Interval Number)



(a) incoming (b) outgoing

Figure 4.8: Packet size distribution for all three gameservers combined

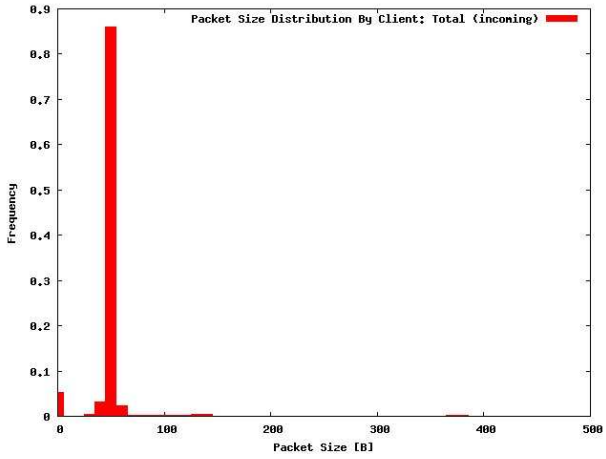
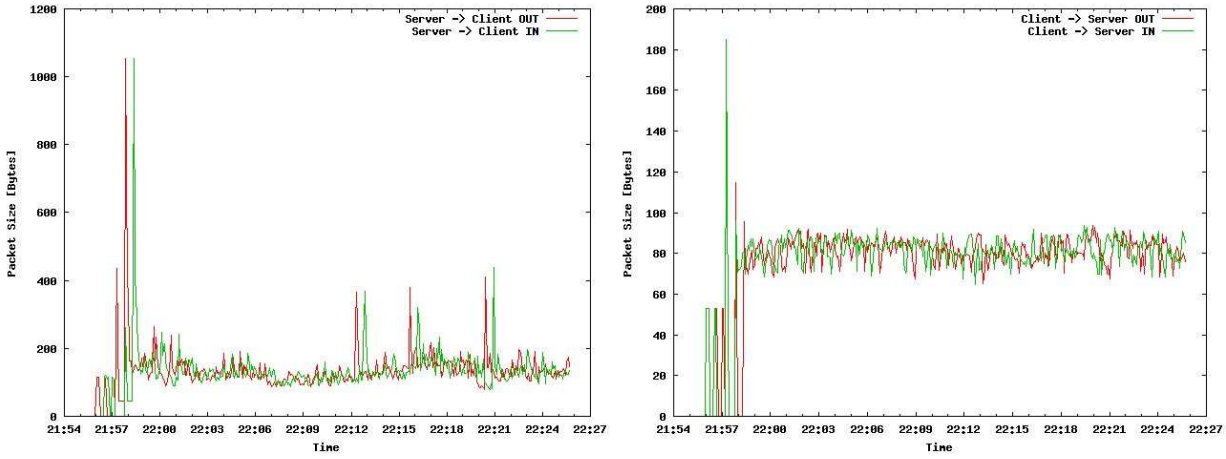


Figure 4.9: Packet size distribution of incoming packets for all three gameservers combined, by client

Figure 4.10 shows that packet sizes may change slightly on the way between client and server, yet as the two system clocks were not synchronized ths may also be an artifact of calculation. Packets sent by the server are generally larger than the ones sent by the client and show more variation.



(a) Server -> Client

(b) Client -> Server

Figure 4.10: Average packet size calculated over 5s intervals for a 30 minute period of high activity on gameserver ETH2

4.2.2 Packet Load

As with packet size, average packet rate (calculated for each 60s period) shows the same periodicity correlating to player activity on the servers, with daily peaks between 1200 and 1600 packets/second and an average of 175.146 packets per second (Figure 4.11).

Figure 4.12 shows average packet rate per 1s interval during two hours of medium server activity distributed more or less evenly over all three servers. Rates are similar and follow similar temporal patterns with the average incoming rate (213.355 packets/second) lower than the outgoing rate (274.926 packets/second) and showing more small-scale variation.

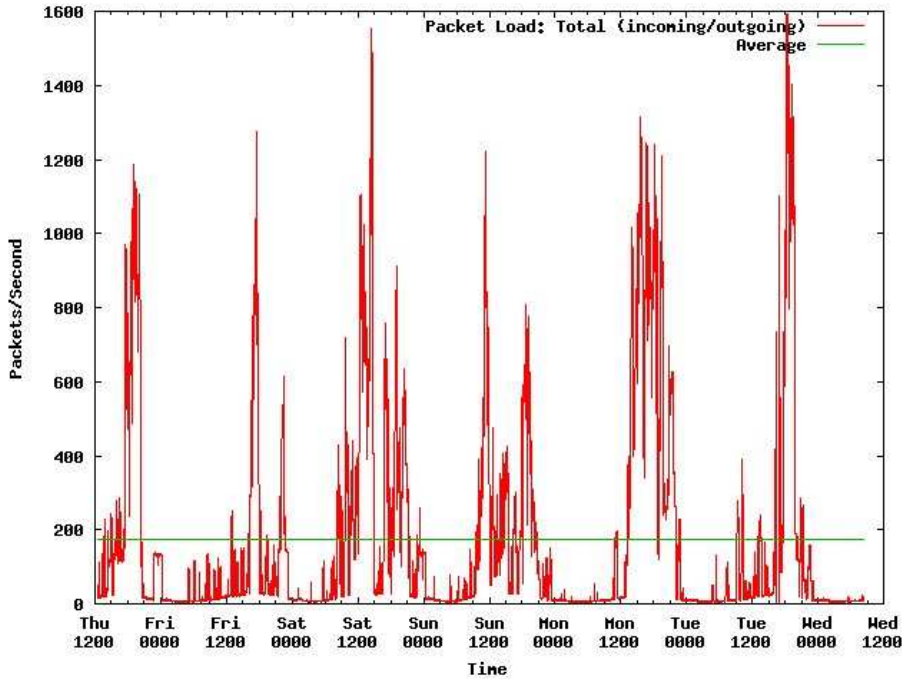
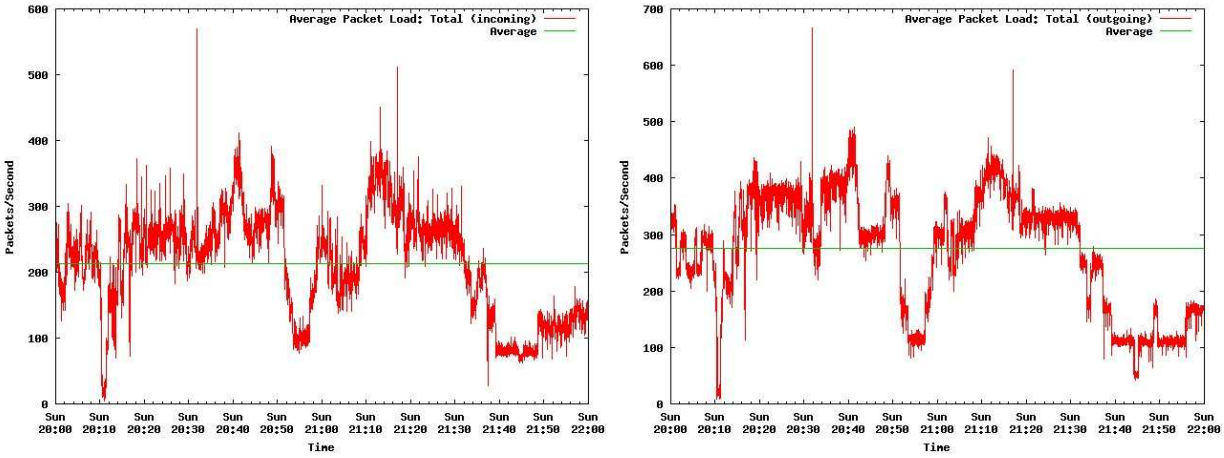


Figure 4.11: Average packet rate calculated over 60s intervals for the total trace duration



(a) incoming (b) outgoing

Figure 4.12: Average packet rate calculated over 5s intervals for a 30 minute period of medium activity on gameserver ETH2

Figure 4.13 shows average packet rates (in/out) calculated over 1s intervals for server 2 with additional markers placed for connection events (arrows over line), disconnect events (arrows below line) and map change events (circles). Player connection and disconnection are clearly the events that influence the packet rate the most, moving the average up or down respectively, yet map changes are also visible if there are sufficient players on the server to raise the average packet rate.

Figure 4.14 shows average packet rates calculated over intervals of 1ms shown for 100ms of activity, thus an average packet rate of 4000 packets/second is caused by four packets arriving within that specific interval of 1ms. The figure shows that the outgoing rate shows clear periodicity, sending few packets roughly every 15ms, whereas the outgoing rate shows no such pattern with individual packets arriving in seemingly random intervals.

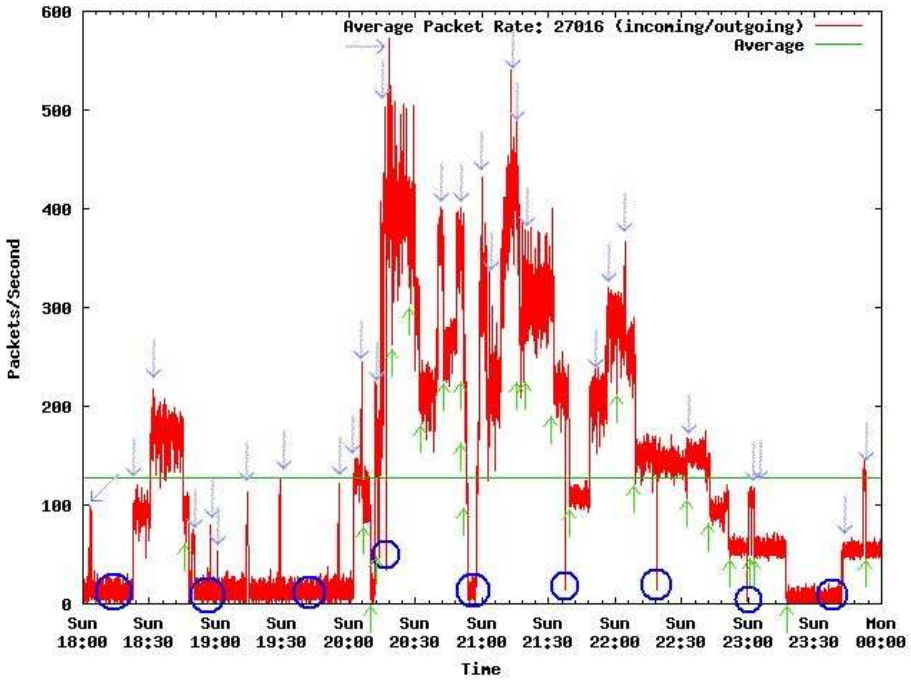
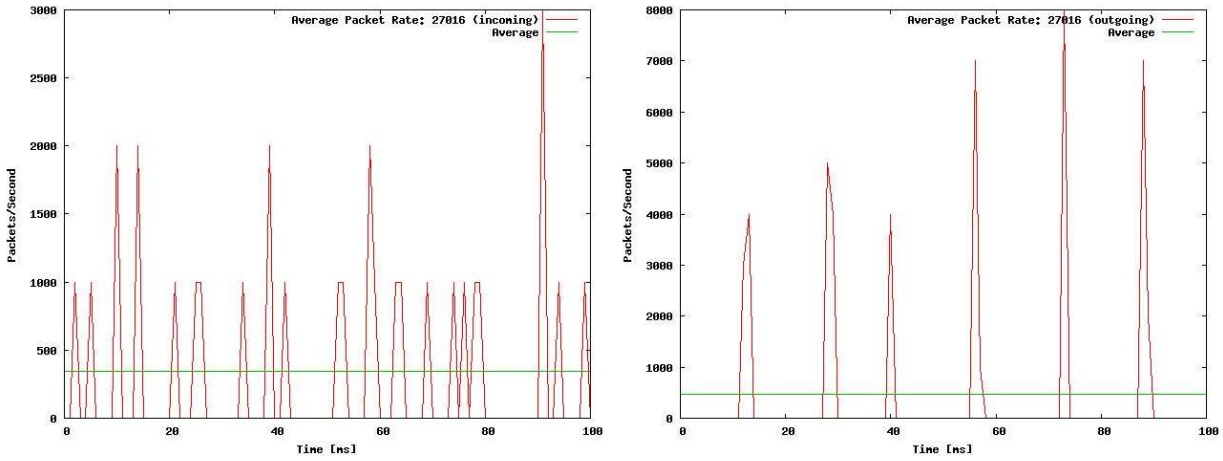


Figure 4.13: Average packet rate calculated over 1s intervals for Sunday evening. Connection (arrows over line), disconnection (arrows under line) and map change events (circles) marked manually

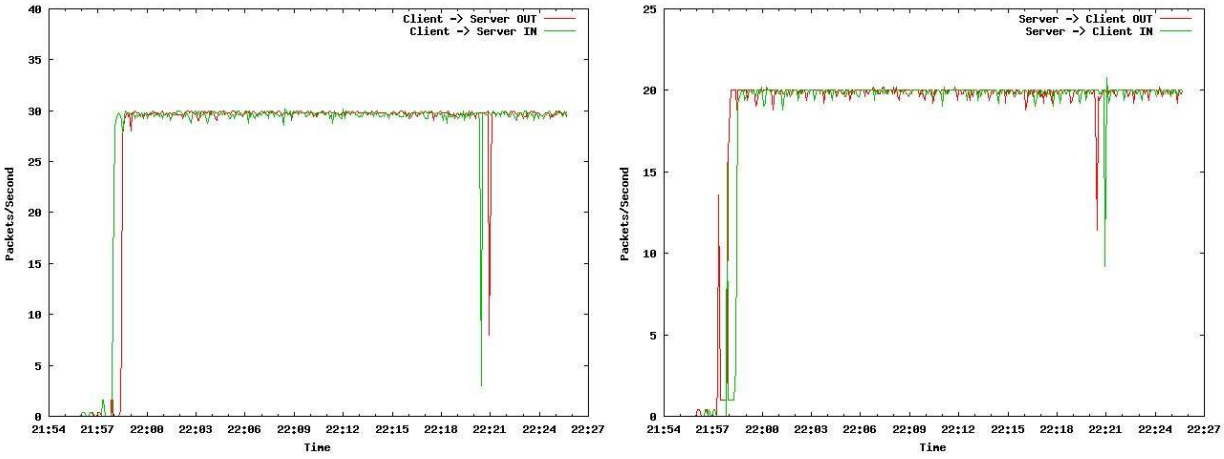


(a) incoming (b) outgoing

Figure 4.14: Average packet rate calculated over 1ms intervals for a 10 second period of high activity on gameserver ETH2

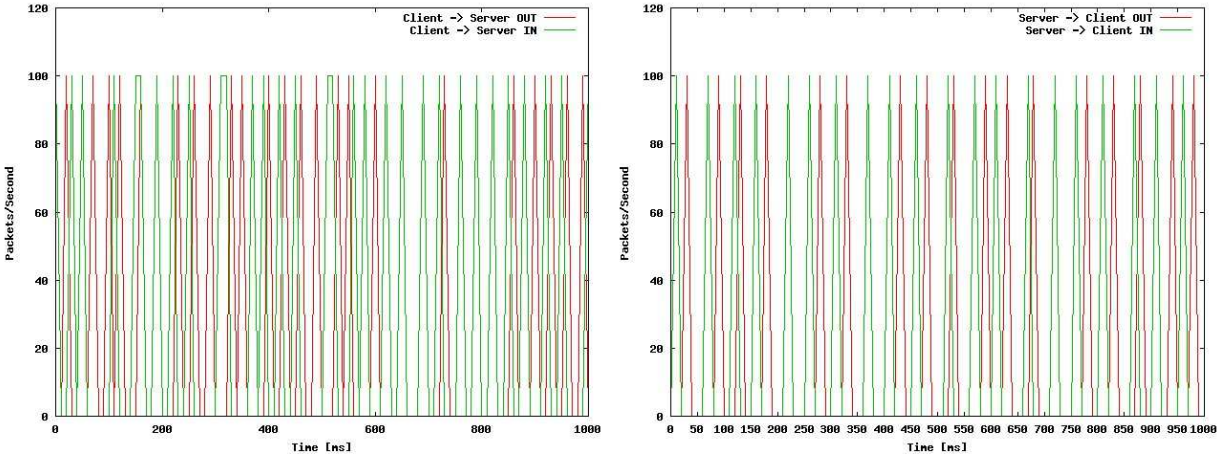
Figure 4.15 again is influenced by unsynchronized system clocks of client and server, yet it shows that packet rates do not seem to suffer major impacts by the network in this setup. Again rates are slightly more variable in the flow from the server to the client, yet both incoming and outgoing traffic show little variation in packet rate once the client is connected and playing. The dip around 22:20 was caused by a map change event. It is also apparent that the client sends packets to the server at a higher rate than vice-versa.

Figure 4.16 shows a detailed view of packet rates calculated every ms over the course of a second. While the lack of clock synchronization will have heavy effects at this level of resolution, the figure shows that traffic from the server to the client is much more periodic and regular than in the other direction.



(a) Client -> Server (b) Server -> Client

Figure 4.15: Average packet rate for flows between client and server ETH2 during approximately 30 minutes of play on a full server



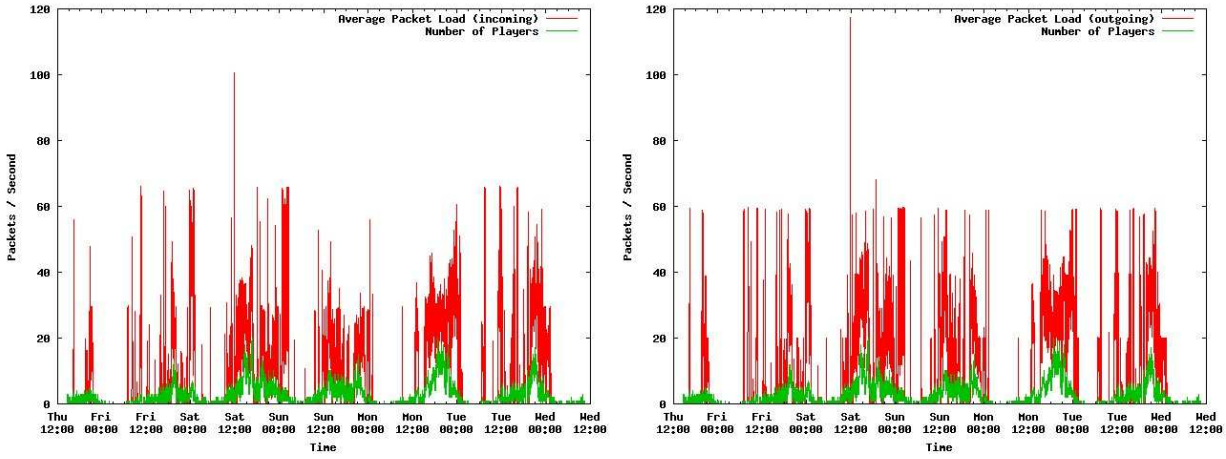
(a) Client -> Server (b) Server -> Client

Figure 4.16: Average packet rate for flows between client and server ETH2 during approximately 1 second of play on a full server

Figure 4.17 shows the number of players connected to all three servers along with the average packet rate of each player's connection to the server. Figure 4.18 shows the number of unique IP addresses communicating with server ETH2 over a 2h period with average packet rate of the "flows" calculated over 1s intervals. Figure 4.19 shows the same information yet taking into account only traffic generated by players connected to the game. Evidently packet rate increases for individual players as the number of connected players increases.

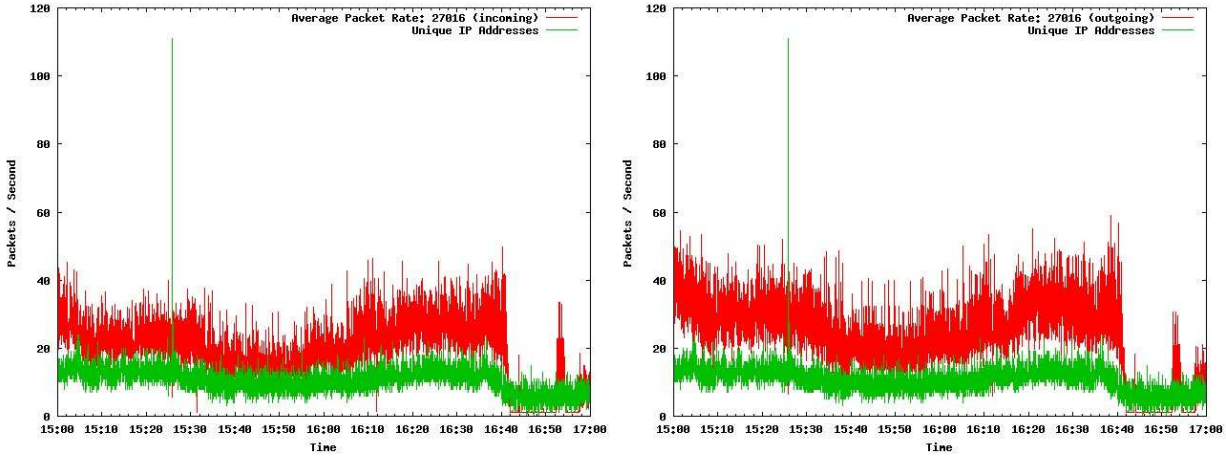
4.2.3 Game-External Traffic

Table 4.2 shows the number of packets generated and received by both the game client and the Teamspeak client. Game traffic made up 89.09% of the incoming and 89.6% of the outgoing traffic in terms of packet numbers. Table 4.3 shows the data produced by the two clients. The



(a) incoming (b) outgoing

Figure 4.17: Average packet rate per player calculated over 60s intervals



(a) incoming (b) outgoing

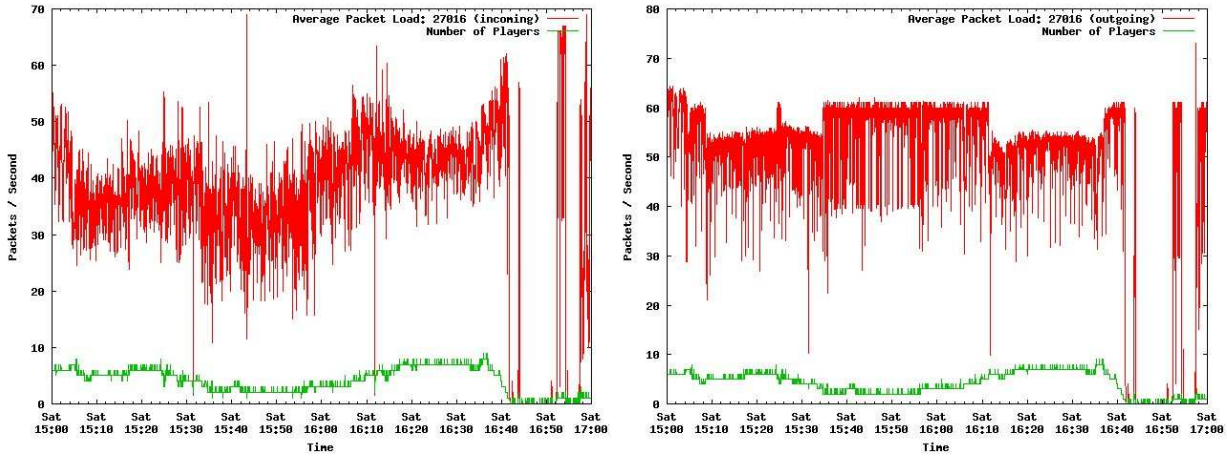
Figure 4.18: Average packet rate per ”flow” calculated over 1s intervals

Counter-Strike: Source traffic made up 91.74% of the incoming and 83.2% of the outgoing traffic. Figure 4.20 shows the average packet sizes of traffic generated by applications external to the actual game but used in parallel, in this case a communication (Voice-Over-IP) tool ”Teamspeak”. The incoming and outgoing Teamspeak traffic look similar, a baseline flow interrupted by bursts of communication activity generating packets of more or less constant sizes.

Figure 4.21 shows average packet rates for the same setup as above. The server produces clearer drops in packet rate when rounds end than the experimental server. The figure also shows that Teamspeak traffic is much less regular than game traffic and produces much less packets than the game client or server.

4.2.4 Pings

Figure 4.22 shows the average ping as provided by querying the game server (SRCDS) and by sending an ICMP ping (ICMP). Both queries were made every 10 seconds and clients that blocked ICMP requests were not considered. The plot shows that ping times were generally

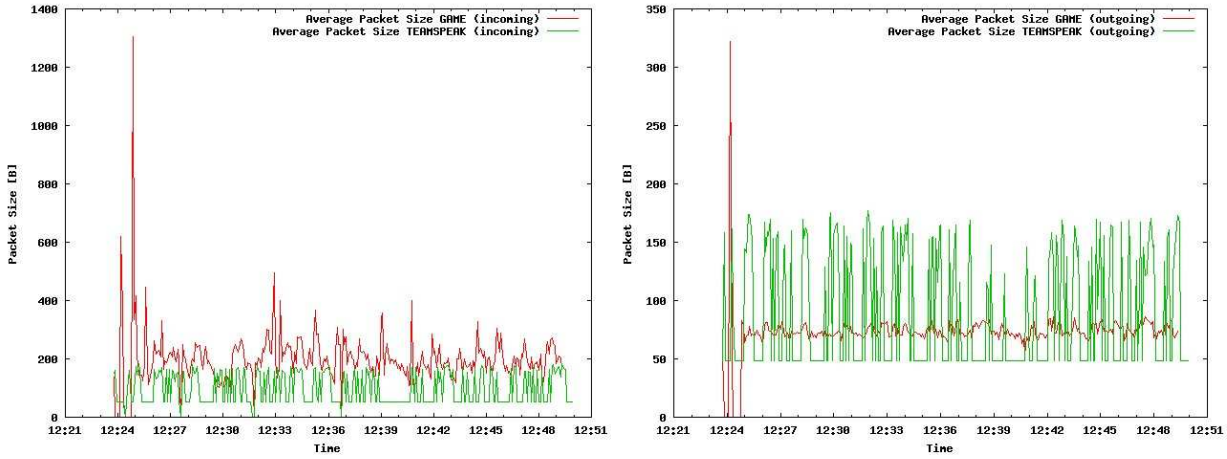


(a) incoming (b) outgoing

Figure 4.19: Average packet rate per ”flow” calculated over 1s intervals

	In	Out	Total
Game	39734	43254	82988
Teamspeak	3820	3559	7379
Total	44601	48275	92876

Table 4.2: Number of packets produced by CSS and Teamspeak client



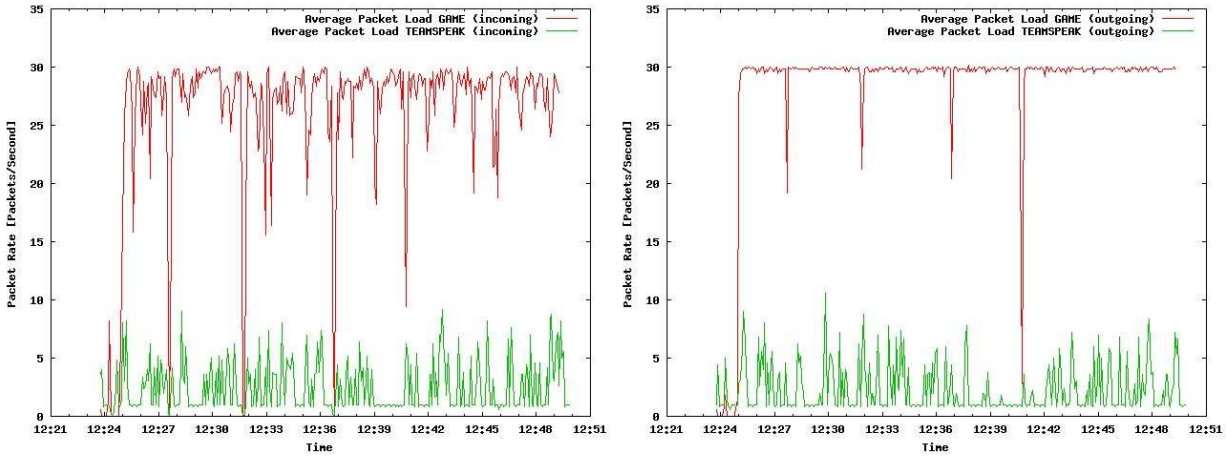
(a) incoming (b) outgoing

Figure 4.20: Average packet sizes over 5s intervals for a Counter-Strike: Source and a Teamspeak client

below 100ms, yet during periods of high activity spikes of high delay can be observed. In the

	In	Out	Total
Game	7810025	3195638	11005663
Teamspeak	533331	459462	992793
Total	8513015	3840837	12353852

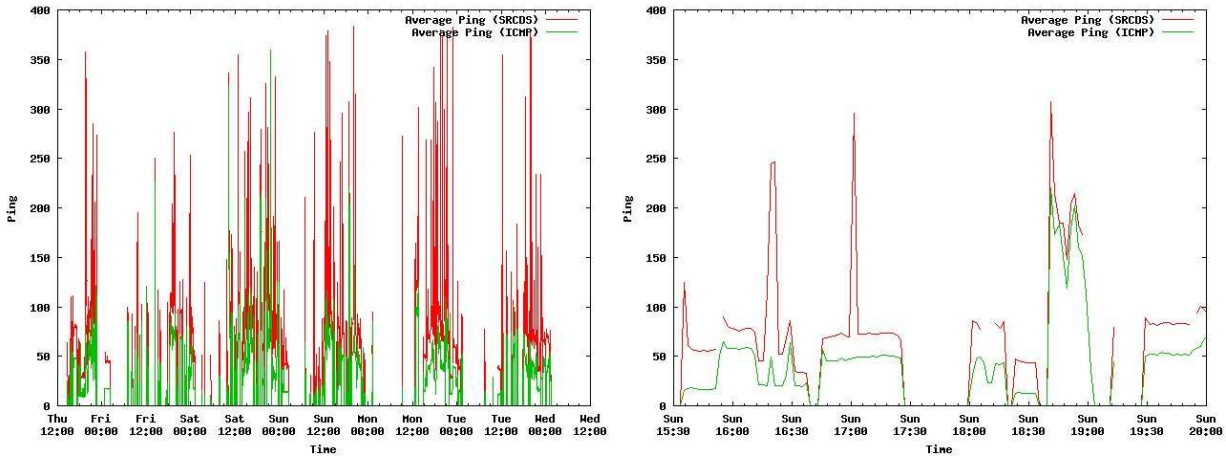
Table 4.3: Data in Bytes produced by CSS and Teamspeak client



(a) incoming (b) outgoing

Figure 4.21: Average packet rates over 5s intervals for a Counter-Strike: Source and a Teamspeak client

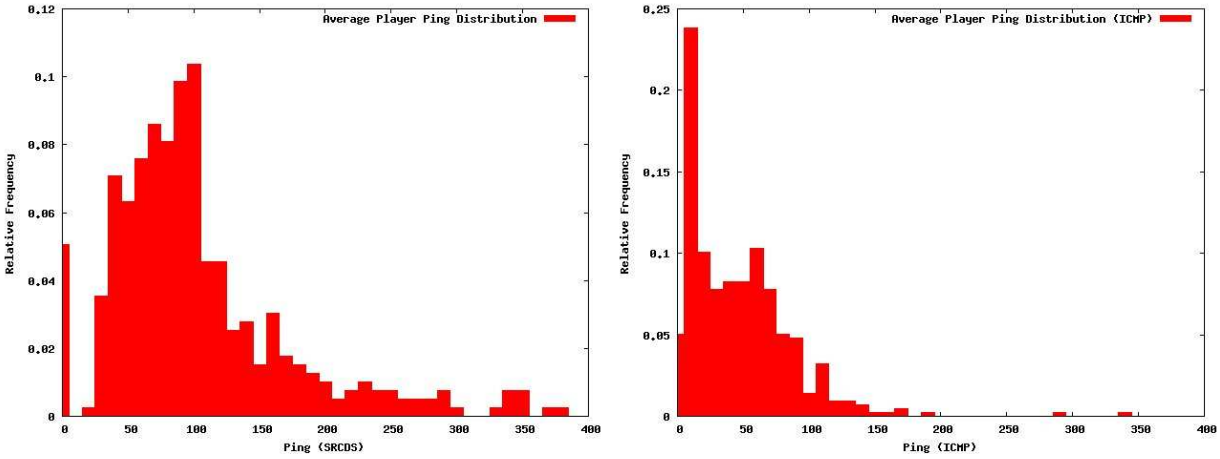
detailed view over a period of relatively high activity all values of 0 and those above 400 were set to ”undefined” as personal observations of pings show that such values seem to be artifacts of the server’s calculations, indeed the ICMP curve tends to be continuous in places where there are breaks in the SRCDS curve. All in all the plot shows that the game server calculates pings that are slightly higher than the ICMP pings (which makes sense as the SRCDS ”ping” includes some time used on packet processing), yet that the overall pattern of the two curves is very similar. Again pings tend to be between 50 and 100ms.



(a) total period (b) detail view

Figure 4.22: Average ping on all three servers both as reported by gameserver (red) and by active ICMP ping request (green)

Figure 4.23 shows the distribution of average player pings over a game session. 56.44% of all players have average pings below 100 (based on SRCDS data). While the distribution does have a heavy tail, the number of players with pings over 100 drops quickly. When looking at the ICMP data the tail is much less heavy and most players have pings below 100ms (83.04%)



(a) SRCDS (b) ICMP

Figure 4.23: Ping distributions by client as reported by the gameservers and ICMP ping request

4.3 Player Behaviour

Overall the players on the players on the servers seemed to be happy with them. On the occasions when the author joined the server to play and test the server behaviour players often commented on the low ping. In addition the author and a fellow clan member regularly played on the servers and monitored player behaviour. Many returning players were noticed that showed up on the server on various occasions, there were even a couple of clans who joined regularly with up to five players. The administrators did not tolerate unfair behaviour and insulting comments and were quick to ban players who violated the unwritten code of conduct. On one occasion a banned player returned after his ban time expired and appologized for his behaviour. Players were often glad to be rid of these unfair and insulting players.
While the servers 1 and 2 were not regularly used some players still joined and played alone against the bots for considerable time. On one occasion the author joined server 1 while there was only one human player on it and that player remained for 20 minutes. Yet based on personal experience players usually prefer to join servers which already have players on them.

4.3.1 Session Times

Figure 4.24 shows the player load on all three servers over the duration of the data sampling period. It shows a clear periodicity as players join the servers around 12pm and again after 6pm until about midnight. Load on Saturday and Sunday was more evenly distributed across the afternoon and evening while the other days show clear spikes. It also shows that server 2 (port 27016) was well-visited while the others were not frequently visited.

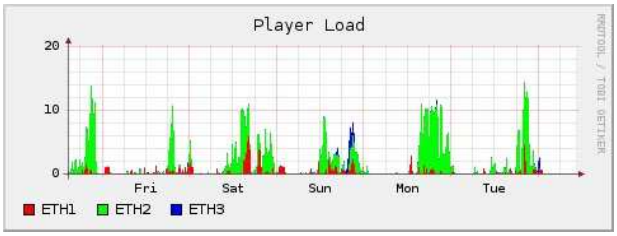
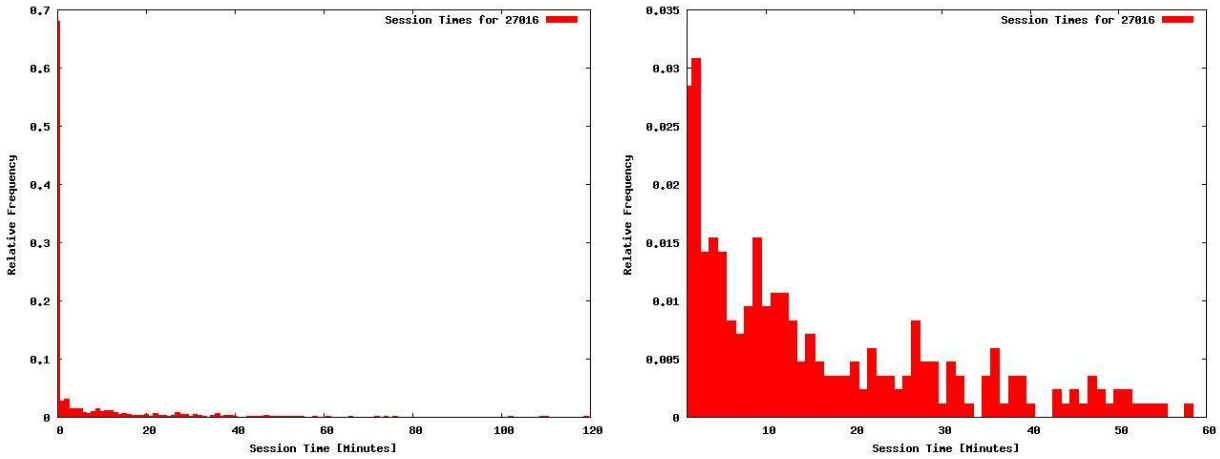


Figure 4.24: Player load on the three gameservers

Figure 4.25 shows the average session times of all players that connected to one of the three

servers during the course of the experiment. It shows that a large number of players connected to the server for only a few seconds before disconnecting again. Yet the distribution has a long tail, with some players playing on the same server for 2h in one session. Figure 4.25(b) shows a detailed view of the times between 1 minute and 1 hour. Short session times are a lot more frequent than longer ones.



(a) Overview (b) Range 1-60 minutes

Figure 4.25: Session time distribution by client session on server ETH2 at two different x-axis ranges

4.3.2 Player Location

Figure 4.26 shows the location of players of all three gameservers. While the underlying map is relatively crude and the resolution of client IP addresses to the coordinates of specific cities using the low-resolution free database may not be very accurate, it still shows that most players come from within continental Europe, with only few players joining from further away such as the Middle-East, the USA or Russia. No players joined from South America, Africa, Oceania or South-East Asia.

4.3.3 Player Load

Figure 4.24 clearly shows that the server on port 27016 (”ETH2: DUST2 Only”) was the favorite server of the three. While all three servers show the same activity periods in the afternoon and evening as well as on weekends, Only server 2 was ever fully loaded with 14 players. In fact server 2 was often fully loaded in the period between 8 and 10pm and that is still the case at the time this document is being written.

4.4 Netflow Data

The following are the results of the NetFlow data analysis. The analysis was done on incoming traffic only.
Figure 4.27 shows the unique IP addresses over 60s intervals of incoming traffic to the gameserver. Figure 4.28 compares the (scaled) number of unique IP addresses making up the incoming traffic to the total number of flows recorded by the DDosVax project over 60s intervals. The total number of flows and the (scaled) number of unique IP addresses sending data to the server show a phase discrepancy, the total number of flows seems to follow office hours

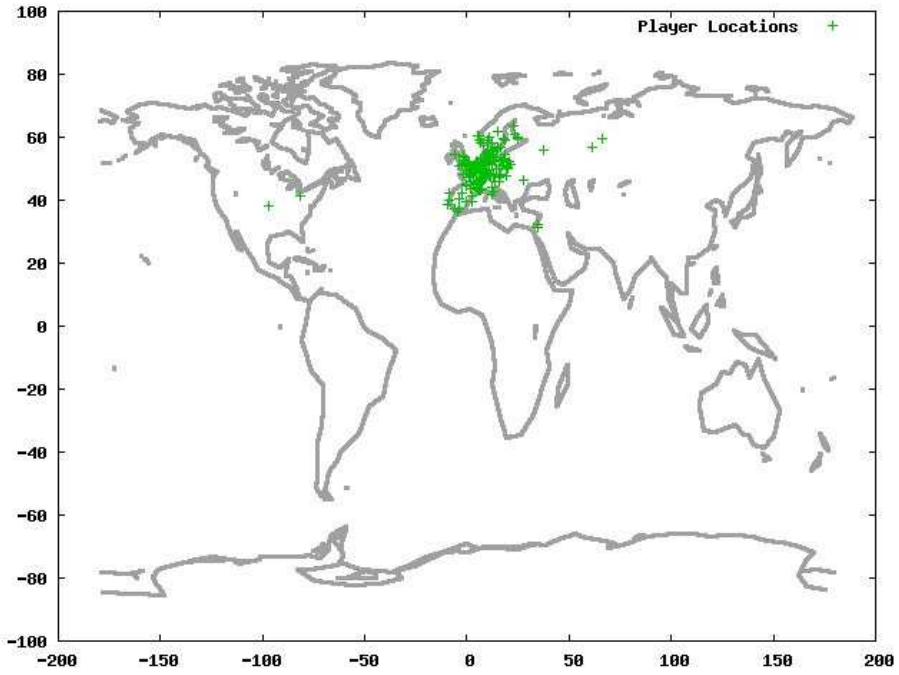


Figure 4.26: Locations of all clients that connected to any of the three servers during the capture period

whereas the total number of unique IP addresses and especially the number of unique IP addresses of clients with rates above 5 packets/second follow the gaming activity periodicity also observed in the packet traces.

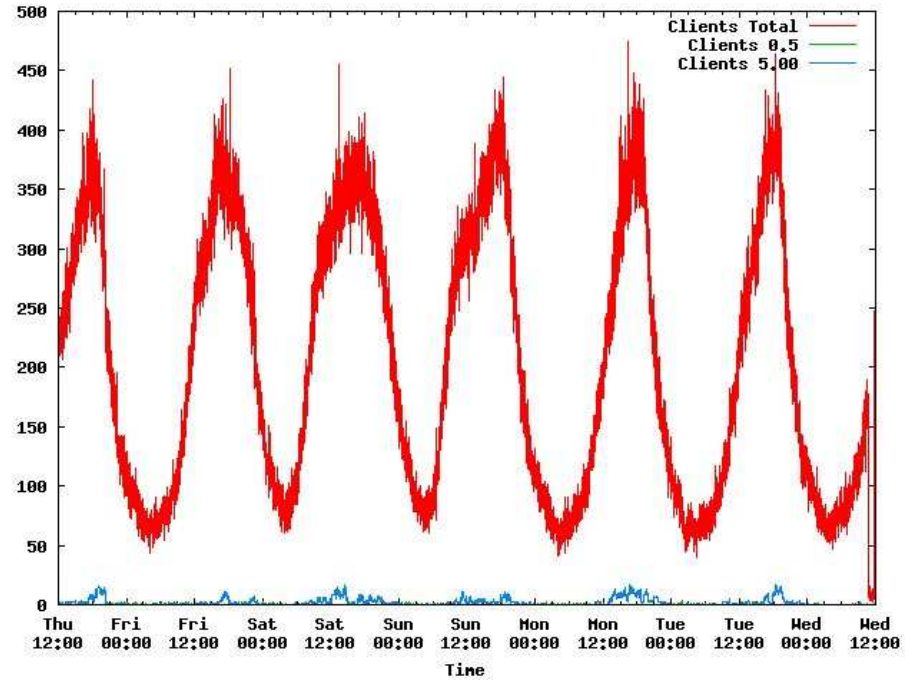
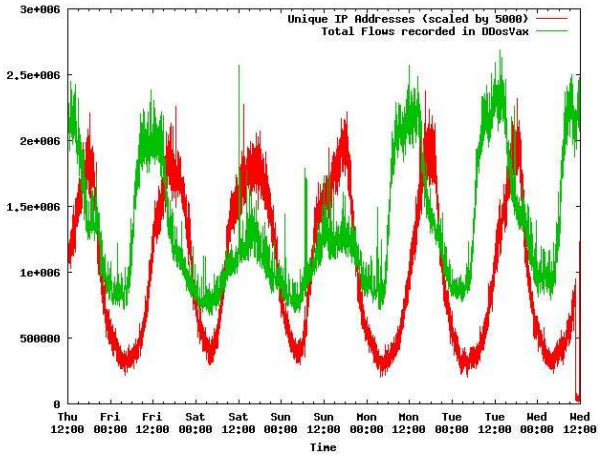
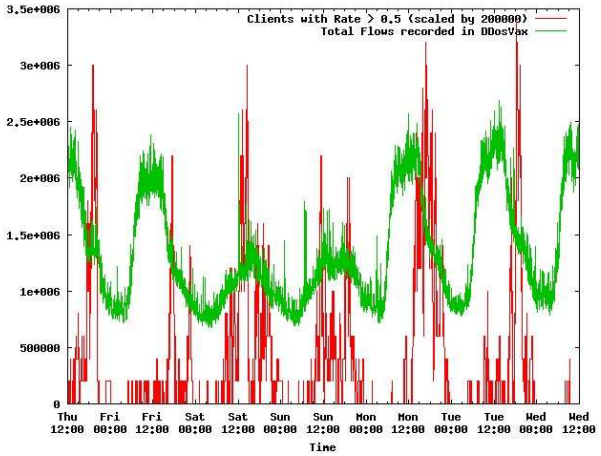


Figure 4.27: Unique IP addresses sending traffic to the gameserver filtered by packet rate r[packets/second]: r>0.00 RED, r>0.50 GREEN, r>5.00 BLUE



(a) Unique IP addresses



(b) Unique IP addresses of clients sending more than 5 packets/second

Figure 4.28: Activity Periods: Total number of flows recorded vs. the scaled incoming traffic to the gameserver

Figure 4.29 shows the average packet size of all incoming traffic to the gameserver calculated over 60s intervals. The same activity periods are visible as in the packet traces, and also the constant influx of 53 Byte status request packets during periods of low to no activity. The low average packet size of 73.411 Bytes is also consistent with the packet trace data.

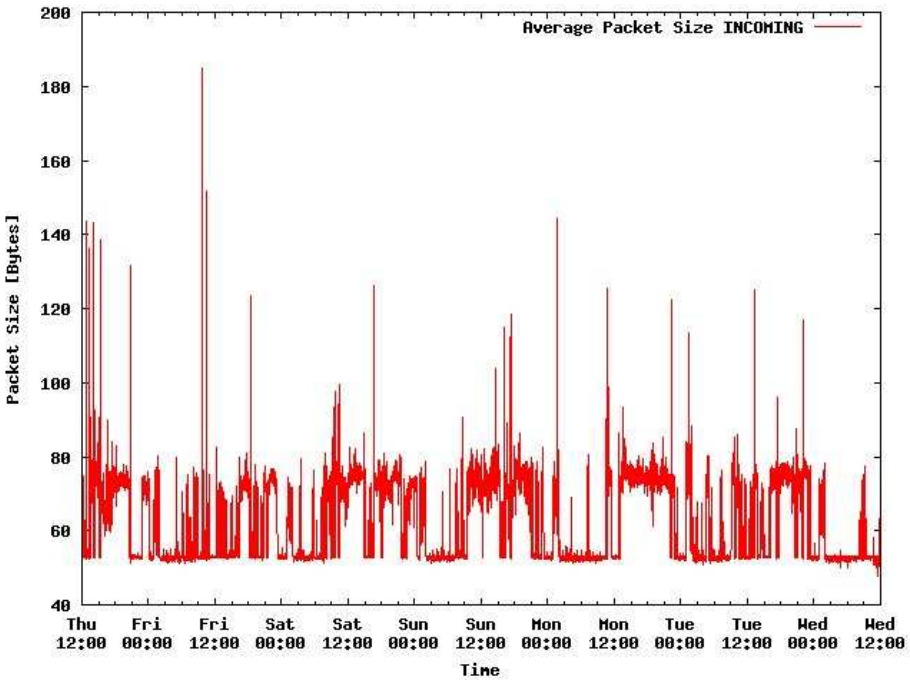
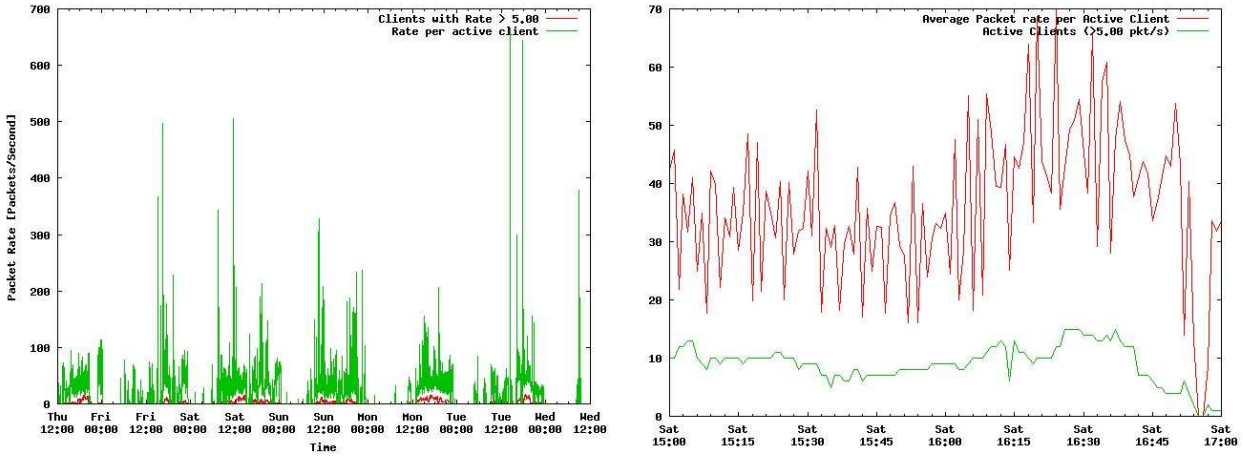


Figure 4.29: Average packet size of total incoming traffic to the server over 60s intervals

Figure 4.30 shows the average packet rate per unique IP address for those clients that have a rate above 5 packets per second. This was done to remove single packet status/information exchanges and tries to capture only active clients actually playing on the server. As with the packet trace data the average rate per connected IP address goes up with increasing player

load (see Figure 4.30(b)) and the average rate is comparable, even though in the trace actually connected players where used whereas in this case a minimum rate cutoff was applied.



(a) Overview (b) Period of medium activity

Figure 4.30: Activity Periods: Total number of flows recorded vs. the scaled incoming traffic to the game-server

Chapter 5

Discussion

5.1 Traffic Characteristics

The first thing that became obvious during the compilation of the results was that there was an enormous variation between the three gameservers due to the fact that of the three servers only one was actually heavily used. The servers were set up to attract as many players as possible by offering a wide range of possibilities to play in addition to having ”ETH” in the server names, which promises low latency connections. While the author was aware of the popularity of the map ”de_dust2” it was a surprise to see that the other two servers, which offered other popular maps, both with and without bots, where often empty while the dust2-only server was full. And as the results show (e.g. Figures 4.2, 4.4 and 4.13) player load affects a gameserver's network characteristics at all scales. Therefor in order to provide useful and representative models of game traffic on a network such inter-server variability caused by client preferences must be taken into account. While this ”player inbalance” influences the overall contribution of each server to the total traffic, the patterns observed coincide with earlier studies even though they were based on single servers ([10]), LAN servers ([8]) or servers with very low (but constant) player load ([9]). However server variability made the aggregate data of all three servers heavily biased towards the second gameserver (”ETH2: De_dust2 only”) and during most of the capture period did not provide a representative picture of what was happening on the other two gameservers. It also made the study less representative of the effects of multiple instances of gameservers of a particular game residing on the same physical machine on each other.

Another important characteristic that showed in the results was the fact that even in ”non-gaming” times the server received (and sent) regular traffic. Closer inspection of the logs and the simultaneous trace on the author's machine and the server showed that this resulted from the client's server browser send a 53 Byte packet to the server requesting the server's status and information, which the server replied to with a 131 Byte status packet.

The study by Feng et al. ([10]) which looked at very similar questions in regards to Counter-Strike (not Source) came up with similar results. While capturing data for a Counter-Strike server in a LAN setting [8] reported an average incoming packet size of 82 bytes and an outgoing average size of 127 Bytes, which is comparable to the packet size averages of this study (4.3). They also showed how outgoing traffic consisted of larger packets than incoming traffic and that outgoing traffic was sent as short bursts. This of course is due to the fact that the network code in the Source generation of the game, while probably improved, was still used to achieve the same goals: the server processes and sends state updates (including information on several players) to each client in cycles, whereas the clients send event information to the server as it happens. Therefor when simulating game traffic two models should be used, one for the server and one for each client. As suggested in [8] this can be done as a stream from each client to the server and a series of bursts from the server to each client. [6] also notes that there will be some variability between clients as computing power (CPU, GPU, RAM, ...) will affect the rate in which packets are sent by the client, as game state calculation and packet creation depend on the available hardware as well as the bandwidth. It also suggests that while

the changes in graphics of computer games over time may be drastic, the main ideas behind the game stay the same, thus servers still only need to calculate positions and send around coordinates, while clients have to take these coordinates and render game figures and textures of map areas, making client side network characteristics more variable over time and more dependent on local hardware than server traffic, which would remain more constant across generations of games. As game development is an expensive process it is also the case that often a company will purchase a game engine and develop their game using that engine, such that a whole ”generation” of FPS games may be based on only a handful of game engines (including the netcode), so that a single model may be able to explain the network traffic caused by a series of games.

Comparing game traffic to background traffic was not possible at this stage of the project as requests for information to ISP's and the ETH network administrators were unsuccessful and no recent studies detailing such issues were found. Based on information from a study done in the year 2000 ([4]) the packets sent to the server from the clients were below the average UDP packet size found on the internet (157 Bytes) and traffic going from the server to the clients also consisted of smaller than average packets in general. The game packet flows seem to be clearly distinguishable from ”regular” IP-based flows which in [4] consist to a very large proportion of packets of three separate sizes: the minimum TCP packet length (no payload) of 40 Bytes, the maximum TCP packet size of 1500 Bytes and packets of either 552 or 576 Bytes from TCP implementations that do not use path MTU (maximum transmission unit) discovery. However since then the proportion of game traffic may well have increased, making detection of distinctive game flows more difficult to discern from background noise, on the other hand, peer-to-peer traffic will most definitely have increased leading to an increase in file transfers that would use large packets.

Previous studies did however differ in their results in regards to certain aspects of player behaviour. In the Feng et al. study ([10]) the server seemed to be constantly fully loaded in terms of players, leading to more regular medium scale patterns of packet load, unlike in Figure 4.13, where players joining and leaving have clear impacts on packet rate, as the server has to send more or less packets within each cycle depending on player load. In addition the results show that the individual packet rate of each connected client increases with increasing player load. The Feng et al. study showed that server events such as map change caused the most significant variability in packet rate, unlike the servers in this study, where players joining and leaving caused much more variability. While their server was located in the US they showed that during times in which players from the US were less likely to join their numbers were compensated by players from Europe or Asia joining the server. Figure 4.26 however clearly shows how the clients on the servers of this study were almost exclusively from within western Europe, while at times in which the european gamers were not interested in gaming the servers remained empty. This is caused by players preferring servers with low pings, i.e. geographically close ones. The Steam platform used to browse for servers actually supports this notion by forcing servers to define their location in their configuration files (”Europe” in the case of the study's servers) and providing filter mechanisms while browsing to display only servers located in the same region.

While game traffic produced by different servers of a particular game could be expected to be similar, the difference seen between Figures 4.15 and 4.21, where in the former the round end events (approximately every 3 minutes) are hard to see while in the latter they produce noticeable drops in packet rate, shows that server settings and the network environment they are in may very well lead to significant variability on the medium scale. It is also possible that the other two servers running on the same machine flattened out the markable drop in packet rate on map change events in the case of the experimental server, yet it is unkown to the author wether the server used in the ”Teamspeak” trace was the sole gameserver on the respective computer. Furthermore Section 4.2.3 shows that while communication traffic is dwarfed by game traffic, it still makes up between 5 and 10% of overall traffic and can thus be expected to have an effect on overall traffic characteristics of a host computer or the actual game server if the voice-over-IP functionality is integrated into the game. The experiment was done with only two people active on the teamspeak channel, a load of 5 to 10 (reasonable numbers for CSS games) or even more

(for games with larger teams, such as MMORPGs or titles in the ”Battlefield” series) will probably increase both the rate and the size of packets. Thus it remains to be seen how such applications affect the traffic footprint of game clients and servers. Again these effects may be skewed by the fact that many online games today provide in-game voice communication, yet personal experience suggests that many players will prefer a controlled communication framework which only an external application can guarantee.

Players select a server to join based on the map being played, the number of players already on the server, ping and previous experience on that server. Latency or ”Ping” is a major factor in server choice, as a player with a high ping is generally at a disadvantage. It has been suggested that players notice a detrimental effect as soon as pings go over 50ms and consider pings above 100 to be bad ([5]). The same study suggests that most players tolerate pings of up to 150ms and that games stay playable with up to 500ms of latency. Personal experience however says that pings between 20 and 50 are considered ok, whereas servers with pings of over 50ms are not joined if there are any alternatives. It is often also the case that servers automatically kick players whose pings rise above 100ms. Overall the impact of latency on a game’s enjoyability depends on the type of game and the networking code of the game, yet FPS games such as Counter-Strike: Source, where players need to show fast reaction to the actions of other players will generally not tolerate much latency. Figure 4.23 however shows that while many players have pings at or below 50ms, there are still many with high pings of up to 150ms, and that the distribution is heavy-tailed. It also shows that the latency calculated using ICMP echo requests from outside the game is often lower than the one reported by the server, which in turn may include some processing time. Personal experience has also shown that pings tend to be high directly after connecting after which they fall to constant low levels. This ”artifact” is also hinted at by Figure 4.22 where pings reported by the gameserver often go to unplayable levels while the ICMP pings stay low around 50ms.

As both packet size and rate have been shown to depend on player load (e.g. [10, 9, 17]) the way players join a server will have a large effect on any network traffic. Thus it is not surprising that those studies show similar patterns for several FPS games where players can join at any time. In contrast in strategy games such as StarCraft players join a game and start a round once the desired number of players has joined, after which the round is played out between those players without other players joining (existing players may drop out if they are defeated). This kind of connection/disconnection pattern may lead to different, less variable traffic patterns than for the more ”dynamic” FPS game type. It has been shown ([9] that while both game types produce small packets, the packet sizes of strategy games indeed tend to be subject to less variation than those of games such as Counter-Strike.

As players can join at any point during the game session times will also affect a game’s long term network characteristics. [7] suggested that a client will produce a more or less constant load on the network once in the ”ON” state (i.e. connected and playing). Figure 4.10 shows that is indeed the case for both packet size and rate for a constant player load on the server. However Figures 4.19, 4.18 and 4.17 suggest that the average packet rate of connected players (and all other flows to and from the server) increases and decreases with the number of connected players (or unique IP addresses).
[7] also shows that many clients are only connected to Counter-Strike servers for a very short amount of time. This is also shown in Figure 4.25. The vast majority of players connect for a very short period. These short sessions can be attributed to players connecting to a server without checking the information available from outside the game and then deciding that they would rather play on a different one. As bots are listed as active players on the overview of the Steam tool (e.g. ”4/14” in the ”Players” column) and can only be identified as bots when clicking on ”Get Server Info” players often connect expecting 4 other people to be already playing on that server only to discover that there are only bots on it, causing an immediate disconnection. While short session times are much more likely than longer ones, there are still a few players who play for up to 2 hours on the same server.

5.2 Gameserver and Client Traffic Patterns

The results showed that a series of patterns exist as possible candidates for a detection mechanism. The feasibility of each of these patterns must of course be demonstrated, yet that is beyond the scope of this study and is left for later studies.

5.2.1 Outgoing (Server) Traffic

Standard ports can still be used as a detection criterion, yet a lot of servers will not be detected using this ”pattern”. As the port number is easily available data it can be used in conjunction with other metrics to confirm a ”hit”, thus if a detection algorithm detects a possible flow that uses the game’s standard port or one close to it, the possible hit can be promoted to a real hit.
Another detected pattern was that outgoing packets were all rather small. The comparison of both the average packet size and the distribution of the packet size with incoming traffic also produces a pattern that can be used for detection. However average packet size and distribution require a certain amount of data to be stored and processed which must not cause congestion and slow downs at the detection site. Thus if a certain IP address sends out a lot of UDP packets within a certain range around a game-specific average the flow could be flagged as a possible game flow.
The fact that servers send out packets in a cycle that looks very regular is also a pattern that should be usable to detect a game server. Besides this ”heartbeat” there is also the map cycle regularity, where the server reloads the map at specific intervals leading to a drop in traffic in both directions at these points in time. Thus a specific IP sending packets at regular intervals could be used to flag a flow as game-related. Of course individual servers use different rates, and these may also differ between games, yet the regularity is definitely an obvious pattern.
One of the most unique patterns observed in this study is the exchange of status requests (53 B) and status information packages (131 B) between the client and the server. Such exchanges were regularly observed throughout the trace. This pattern seems to be a ”fingerprint” of the game and should be detectable, especially at times of low game activity where such interaction produced the majority of traffic in this study. If other games that provide status request functionality to clients have similar but distinct ”fingerprints” this pattern should make it easy to detect and categorize game flows.
As most game servers are online for extended periods of time it would also be possible to make use of the activity patterns of the servers over longer periods (e.g. a week) to classify a flow as being caused by a game server. As the results show game servers are used by geographically close clients mostly outside office hours and the peak of unique connected clients is outside that of ”normal” traffic. This of course affects outgoing packet rate and size.

5.2.2 Incoming Traffic

As with outgoing traffic the incoming traffic is made up almost exclusively of UDP packets of small sizes and an even smaller variability than observed in the outgoing traffic. This pattern seems to be useful for detection purposes. Thus if an IP address receives a lot of UDP packets in a very narrow size range that traffic could be flagged as game related.
The rate of incoming packets is more or less random as players connect and disconnect at random and show host-dependent variability, but the packet rate per individual client increases with the number of ”active” clients, which could be used if there are enough resources available to the detection mechanism to store client states and process the flows on a per client basis (unique IP address). As with outgoing traffic the peak in the number of active clients (and unique IP addresses in general) is outside office hours and could also be used for detection.
Another pattern is that of low latency and geographic closeness. A detection system could examine all flows with longer duration (e.g. 5 minutes) for latency and geographic location, if latency of most connections to a unique IP address are below a certain threshold (e.g. 75ms) and a lookup shows that the source of the connection is close to the destination, then the two IP addresses could be flagged as possibly belonging to a client/server pair for further analysis. Of course this would require the detection framework to perform lookups and possibly perform active latency tests on the IP addresses, which may not be feasible given the amount of traffic passing by such a detection system and the available resources. In addition active latency tests

would cause more network traffic, making this approach much less likely to be implementable. However, if resources available to a monitoring and detection tool are a limiting factor, i.e. the monitor is hard pressed to manage large data structures for IP addresses and calculating statistics in between the arrival of two NetFlow packets, it may be useful to use geographic proximity to filter out flows between two hosts that are far apart.

5.2.3 Netflow Data

Overall there are a series of patterns that distinguish traffic created by a Counter-Strike: Source server covering a series of metrics, ranging from packet size and rate statistics over specific packet exchange patterns through geographic location and latency and activity periods. The results also show that many of these patterns can also be observed without access to the gameserver or client by using NetFlow data. The status-request fingerprint, average packet sizes, changes in per client packet rate and activity period patterns were confirmed using NetFlow data made available by the DDosVax project ([22]) and using a data retrieval, sorting and processing framework that is currently being developed by Dominik Schatzmann ([34]). However, due to the aggregation of packets to flows packet level information based on the analysis of traffic over very short periods of time such as the server clock or tickrate will not be visible in NetFlow records. What remains to be seen is if using these patterns ”in the wild” it will be possible to detect game flows without any prior knowledge, in particular if the resources required for the computations and comparisons are reasonable.

Chapter 6

Conclusion, Contribution and Outlook

6.1 Conclusion

Packets of three Counter-Strike: Source servers running on the same machine within the ETH network and with public access over the internet were captured over the course of approximately six days. Network characteristics are similar to those of other FPS games including the first Counter-Strike (e.g. [10, 9, 8, 16]). It was shown that incoming traffic consists of a stream of small packets, while outgoing packets are slightly larger and are emitted in predictable bursts. It was shown that connection and disconnection of players have a major impact on packet rates. Thus models should include the human factor, such as server popularity, session times (including ”reconnaissance” connections), geographic location and time of day.

As game traffic will most probably continue to increase in importance and gameservers are often aggregated at hosting companies such models may be used to develop QoS functionalities to optimize hardware and protocols for game traffic at such local ”hotspots”. In addition models of game traffic may have to be incorporated into simulators of network traffic to make such simulations more representative of real internet traffic.

It has also been shown that based on characteristics as the ones displayed by the servers in this study it is possible to detect and classify game flows in real-time ([19]). As network characteristics seem to have commonalities among games and even seem to be very similar between two generations of the same game it may be possible to use such detection procedures over longer periods of time and across computer game generations. Especially the exchange of status request packets and server information packets between the server and the clients' server browser promises to be a quick and easy way to detct the presence and address of a Counter-Strike: Source server. Indeed if such ”fingerprints” exist for other games it may provide a quick way to detect and classify servers of a range of games. The narrow distribution of incoming packet size (client->server) and the larger and more variable outgoing packet size also seems to be a good way to detect such traffic. The study clearly shoed that the clients' packet rate increases with the number of connected clients, however using such a pattern to detect game-related traffic requires the monitor to store and process large (depending on network size) data structures (e.g. a hashtable with the source address of a flow as key). In such cases the available resources might limit the usefulness of such detection patterns, although using geographic proximity and low-latency to filter flows may reduce the amount of data that has to be processed. Overall simple patterns such as the ”status request fingerprint” should be much more useful, especially as they possibly allow not only the detection but also the classification of game-related traffic to a specific game. This study also showed that patterns observed using packet traces at the server can also be picked up using NetFlow data, it remains to be seen if an online detection framework has enough resources to detect and classify flows ”in the wild”.

6.2 Contribution

Three Counter-Strike: Source gameservers have been set up along with the tools required to monitor network traffic, player load and CPU load online. It was shown that the presence of these tools does not hamper gameplay on the servers and that processing power and available bandwidth suffice for the lag-free operation of the gameservers. While servers ETH1 and ETH3 still do not attract that many players, ETH2 is often fully loaded on evenings and weekends, with many returning players. The other two servers also show some increase in popularity. Overall the servers can now be used for further analyses of CS:S related traffic.
Two prime-candidates for the detection of network traffic generated by (or sent to) a Counter-Strike: Source server have been identified that are not only visible in packet traces but also in aggregated NetFlow data: the ”Status Request Fingerprint” and the narrowly distributed size of incoming packets (client->server) coupled with the larger and more variable outgoing packet size.

6.3 Outlook

What remains to be seen is how the new genre of very popular Massively Multiplayer Online (Roleplaying) Games differs in regards to network traffic characteristics to games such as Counter-Strike: Source. While players can leave and join at their discretion in both cases, MMORPGs will probably have longer session times and many more players per server than Counter-Strike: Source. In addition the servers used for MMORPGs are hosted by the company that produced the game, must be big enough to host hundreds of players playing simultaneously and are professionaly managed and kept up-to-date and running, whereas FPS servers are often free for use and are either hosted by specialized gameserver hosting providers or even by private persons with many possibilities of individual configuration. There are most probably also far less servers per MMORPG title than per FPS title, at least for the most popular FPS games. This leads to a greater inter-server variability for FPS traffic ”generators”. Therefor the characteristics of MMORPGs must be individually assessed.
Peer-to-peer systems have also started to appear in many application domains. It remains to be seen if peer-to-peer technology is ever adopted for games such as Counter-Strike: Source, as players must trust the server to truthfully process and deliver game state updates, whereas players may not trust other players to compute game state. Such distributed systems may make cheating even more of a problem than it already is. Yet if peer-to-peer systems do appear and become popular, the characteristics shown in this and similar studies for client-server systems will no longer hold for such new peer-to-peer online games and models will have to be adjusted, expanded or replaced.
Based on the sparse information available on overall characteristics of internet traffic and the fact that the results in this experiment confirm those of similar previous studies it can be assumed that game traffic flow can indeed be detected in near real-time based on statistics of packet size and rate. If this is the case for all types of games remains to be seen, indeed for a decisive answer a detection algorithm must be implemented and tested in a system like UPFrame and used to detect such flows among ”real” internet traffic. Once a flow has been detected based on small scale patterns it may be possible to reevaluate the classification on medium term patterns of the flow if it is caused by an actual game session. The detection algorithm would then probably have to be adapted to recognize the flows of different game styles. In addition the status-request/status-information exchange between the client and the gameserver as shown in this study for Counter-Strike: Source seems to be a quick and easy way to detect the presence of a gameserver in a network, what remains to be tested is if such a fingerprint exists for other games and if an online monitoring framework can successfully pick up such exchanges and correctly classify an IP as belonging to a gameserver, and if false positives are minimized.

Appendix A

Game Server Setup and Configuration Files

A.1 Setting Up a CS:S Server (Linux)

The information in this section is based on the more detailed tutorial found at [35]. This section gives a quick overview of how to install a Counter-Strike: Source server on a Linux machine. This may have to be donw with root privileges.

- **Step 1, Get HLDS:** HLDS (Half-Life Dedicated Server) is the actual server software required. Simply change to the directory in which you want to install the server and use the following command:

  ```
  wget http://www.cstrike-planet.com/dls/hldsupdatetool.bin
  ```

  Once the download is complete you simply execute hldsupdate.bin (permissions may have to be adjusted). After agreeing to the ”Terms and Conditions” the tool should produce a new file called ”steam”.

- **Step 2, Install HLDS:** In order to install a Counter-Strike: Source dedicated server, run the following command:

  ```
  steam -command update -game "Counter-Strike Source" -dir .
  ```

  (note the ”.” after -dir). This should start the download of the server files from the Steam network, which lasts between 30-60 minutes depending on network connection and the load on the Steam network. When it is complete it will dislpay a "HLDS installation up to date" message.

- **Step 3, Run HLDS:** The download has now produced a file ”steam_run”, which is used to start the server. The following starts the ETH1 server used in this thesis with a maximum of 14 players, the map ”cs_italy” on the IP address of the server and port 27015 with a tickrate of 66 (sends out updates 66 times per second) and in autoupdate mode (checks for available updates on start):

  ```
  srcds_run -game cstrike +maxplayers 14 +map cs_italy +ip 82.130.102.202
   +port 27015 -tickrate 66 -autoupdate
  ```

A.2 Adding Mani Admin Mod to the Server (Linux)

This information is taken from [23]. Simply get the newest plugin version from [23] and follow the installation instructions available on that site. The following is the main installation page take from the official site, read there for more details on configuration.

Mani Admin Plug-in for Counter Strike Source & HL2 Deathmatch
Installation (Linux & Windows)
Creating the required directories

It is assumed that the base directory of your css/hl2dm installation contains the following directories:

$BASE = path to the directory where the following directories are
$BASE/bin/
$BASE/cstrike/
$BASE/hl2mp/

For CS Source: -

1. Create a new directory $BASE/cstrike/addons/
2. Create a new directory $BASE/cstrike/cfg/mani_admin_plugin/
3. Create a new directory $BASE/cstrike/cfg/mani_admin_plugin/language/
4. Create a new directory $BASE/cstrike/cfg/mani_admin_plugin/map_config/
5. Create a new directory $BASE/cstrike/cfg/mani_admin_plugin/restrict/
6. Create a new directory $BASE/cstrike/cfg/mani_admin_plugin/mani_logs/
7. Create a new directory $BASE/cstrike/sound/admin_plugin/
8. Create a new directory $BASE/cstrike/sound/admin_plugin/actions/

For HL2 Deathmatch: -

1. Create a new directory $BASE/hl2mp/addons/
2. Create a new directory $BASE/hl2mp/cfg/mani_admin_plugin/
3. Create a new directory $BASE/hl2mp/cfg/mani_admin_plugin/language/
4. Create a new directory $BASE/hl2mp/cfg/mani_admin_plugin/map_config/
5. Create a new directory $BASE/hl2mp/cfg/mani_admin_plugin/mani_logs/
6. Create a new directory $BASE/hl2mp/sound/admin_plugin/
7. Create a new directory $BASE/hl2mp/sound/admin_plugin/actions/

Assuming you are using ftp. A good free ftp client can be found at http://filezilla.sourceforge.net/

NOTE : If you are installing for HL2 Deathmatch, your installation directory is hl2mp not cstrike

From the zip file copy the following default files to $BASE/cstrike/cfg/mani_admin_plugin/

actionsoundlist.txt
admingroups.txt
adminlist.txt
adverts.txt
cexeclist_all.txt
cexeclist_ct.txt
cexeclist_player.txt
cexeclist_spec.txt
cexeclist_t.txt
commandlist.txt
crontablist.txt
default_weapon_restrict.txt
gametypes.txt
gimpphrase.txt
immunitygroups.txt
immunitylist.txt
pingimmunity.txt
quakesoundlist.txt
rconlist.txt
reserveslots.txt
restricted_weapons.txt
soundlist.txt
votequestionlist.txt
voterconlist.txt
webshortcutlist.txt
wordfilter.txt

From the zip file copy the following default files to $BASE/cstrike/cfg/mani_admin_plugin/language/

english.cfg
language.cfg

From the zip file copy the following default file to $BASE/cstrike/cfg/mani_admin_plugin/map_config/

cs_reflex3.cfg

```
From the zip file copy the following default file to $BASE/cstrike/cfg/
Note if you already have an autoexec.cfg file in $BASE/cstrike/cfg/ you must manually copy and paste
 the contents of the autoexec.cfg from the zip file into your existing one on the server.

mani_server.cfg autoexec.cfg
From the zip file copy the following default file to $BASE/cstrike/cfg/mani_admin_plugin/restrict/

cs_reflex3_restrict.txt

From the zip file copy the following files to $BASE/cstrike/addons/

mani_admin_plugin.vdf

For linux users: -
mani_admin_plugin_i486.so

For Windows Users: -
mani_admin_plugin.dll

More in the soundlist.txt section on how to do this.
```

A.3 mani_server.cfg

The following is the configuration file for the mani admin modification of the CS:S servers.

```
// ****************************************************************************
// Plugin : Mani Admin Plugin
//
// Filename : mani_server.cfg
//
// Last updated : 03/12/2006
//
// Desc : This is the core configuration file to turn on/off functionality
//        and to also config options within the plugin. By default the main
//        functions are turned off. For the most part the on/off switch is
//        usually the first cvar under each module (i.e mani_adverts,
//        mani_stats, etc)
//
//        In order for this config to be used by the plugin an exec command
//        must be placed in the server.cfg file within the /cfg folder.
//        'exec mani_server.cfg'
// ****************************************************************************

echo "******     Executing mani_server.cfg      ******"

// ****************************************************************************
// Module : Advert
//
// Desc : Adverts can be used in game using this module in conjuction with the
//        adverts.txt file in the /cfg/mani_admin_plugin/ folder
// ****************************************************************************

// Adverts 1 = on, 0 = off
mani_adverts 0

// Time between adverts displayed
mani_time_between_adverts 120

// Allow adverts in chat area of screen
mani_adverts_chat_area 1

// Allow adverts in top left corner of screen (these can be coloured)
mani_adverts_top_left 1

// Sets colour of adverts, here I've set it to blue

// Red component colour of adverts (255 = max)
```

```
mani_advert_col_red 0

// Green component colour of adverts (255 = max)
mani_advert_col_green 0

// Blue component colour of adverts (255 = max)
mani_advert_col_blue 255

// This settings allows you to specify whether everyone can see adverts
// or only dead players. 0 = All players whether dead or alive, 1 = Dead only
mani_advert_dead_only 0

// Show adverts in the hint text area (at the bottom middle in CSS)
// 1 = enabled, 0 = disabled
mani_adverts_bottom_area 1

// ****************************************************************************
// Module : Stats
//
// Desc : The stats module is a very simple ranking system based on 3 methods of
//        calculation (defined in mani_stats_calculate).
// ****************************************************************************

// 1 = Enable stats module, 0 = disable stats module
mani_stats 0

// 0 = calculate once per map, 1 = calculate at end of each round (CSS Only)
mani_stats_mode 1

// Number of days since player last connected before they are removed from the
// stats list
mani_stats_drop_player_days 5

// This cvar is used to set the type of stats calculation to use for ranking
// players
// 0 = Rank by by pure kills
// 1 = Rank by by kill:death ratio
// 2 = Rank by kills minus deaths
// 3 = Rank by points (points delta = (victim_points/killer_points) * multiplier
mani_stats_calculate 3

// Number of kills required before a player is given a rank
// If you are using the Kill Death ratio you should set this quite high
mani_stats_kills_required 0

// Number of kills + deaths required before a victims points are affected by the
// attackers kills. This prevents new players from affecting regular players points
// until a certain amount of experience is gained from playing.
mani_stats_kills_before_points_removed 0

// Defines how long a 'top' display lasts for before it fades (5 - 30 seconds)
mani_stats_top_display_time 10

// Defines whether other players see your rank when you type 'rank'
// 1 = show rank to all players
// 0 = only show rank to player who typed 'rank'
mani_stats_show_rank_to_all 1

// Defines a message to show when a user types 'rank' and the stats are turned
// off (this can be blank)
// mani_stats_alternative_rank_message "www.mystats.com"

// Enables writing of ranks to a text file called mani_ranks.txt for export to a
// web page.
mani_stats_write_text_file 1

// Set in minutes how often you want the stats to recalculate. This should be
// used if you have long map times with no end of round.
// 0 = disables frequency calculating, > 0 = time in minutes between each stats
// rank calculation
mani_stats_calculate_frequency 0

// Set in minutes how often you want the stats to recalculate AND write to disk
```

```
// This should be used if you have long map times with no end of round.
// 0 = disabled, > 0 = time in minutes between each save and recalculation of
// ranks
mani_stats_write_frequency_to_disk 0

// Set to 1 if you want your ranks to be by steam id (default),
// Set to 0 if you are not using steam ids on your server (Lan mode)
mani_stats_by_steam_id 1

// 1 = Include any bot kills made in stats
// 0 = Killing a bot does not count to stats
mani_stats_include_bot_kills 0

// Stats points decay settings
// Number of days since last connect that points decay will start
mani_stats_decay_start 2

// Number of days that the decay will take place over once started
// Points will drop to 500 over this period of time. If the player rejoins
// their points will be restored to full value
mani_stats_decay_period 7

// When a player reconnects the stats module can restore a players
// points back to the full amount if decay has set in
// 0 = Do not restore full points, 1 = restore to full points
mani_stats_decay_restore_points_on_connect 0

// If set to 1 a victim will never lose points ala BF2
mani_stats_points_add_only 0

// Number of days before a player is made invisible from
// being ranked. Note that the player is not dropped, if
// they rejoin their rank will be restored.
mani_stats_ignore_ranks_after_x_days 21

// Multiplier used in points calculation, default is 5.0
mani_stats_points_multiplier "5.0"

// Multiplier for victim points lost. If you want victims
// to lose say half points for dying set this to "0.5" etc
mani_stats_points_death_multiplier "1.0"

// Weapon weighting for CSS Stats
// Making a weight 2.0 will double the points given/taken
// Making a weight 0.5 will halve the points given/taken
mani_stats_css_weapon_ak47 "1.0"
mani_stats_css_weapon_m4a1 "1.0"
mani_stats_css_weapon_mp5navy "1.2"
mani_stats_css_weapon_awp "1.0"
mani_stats_css_weapon_usp "1.4"
mani_stats_css_weapon_deagle "1.2"
mani_stats_css_weapon_aug "1.0"
mani_stats_css_weapon_hegrenade "1.8"
mani_stats_css_weapon_xm1014 "1.1"
mani_stats_css_weapon_knife "2.0"
mani_stats_css_weapon_g3sg1 "0.8"
mani_stats_css_weapon_sg550 "0.8"
mani_stats_css_weapon_galil "1.1"
mani_stats_css_weapon_m3 "1.2"
mani_stats_css_weapon_scout "1.1"
mani_stats_css_weapon_sg552 "1.0"
mani_stats_css_weapon_famas "1.0"
mani_stats_css_weapon_glock "1.4"
mani_stats_css_weapon_tmp "1.5"
mani_stats_css_weapon_ump45 "1.2"
mani_stats_css_weapon_p90 "1.2"
mani_stats_css_weapon_m249 "1.2"
mani_stats_css_weapon_elite "1.4"
mani_stats_css_weapon_mac10 "1.5"
mani_stats_css_weapon_fiveseven "1.5"
mani_stats_css_weapon_p228 "1.5"
mani_stats_css_weapon_flashbang "5.0"
mani_stats_css_weapon_smokegrenade "5.0"
```

```
// Bonus Points for CSS Players
mani_stats_css_bomb_planted_bonus "10"
mani_stats_css_bomb_defused_bonus "10"
mani_stats_css_hostage_rescued_bonus "5"
mani_stats_css_hostage_killed_bonus "-15"
mani_stats_css_vip_escape_bonus "4"
mani_stats_css_vip_killed_bonus "10"

// Bonus Points for CSS Teams
mani_stats_css_ct_eliminated_team_bonus "2"
mani_stats_css_t_eliminated_team_bonus "2"
mani_stats_css_ct_vip_escaped_team_bonus "10"
mani_stats_css_t_vip_assassinated_team_bonus "6"
mani_stats_css_t_target_bombed_team_bonus "5"
mani_stats_css_ct_all_hostages_rescued_team_bonus "10"
mani_stats_css_ct_bomb_defused_team_bonus "5"
mani_stats_css_ct_hostage_killed_team_bonus "1"
mani_stats_css_ct_hostage_rescued_team_bonus "1"
mani_stats_css_t_bomb_planted_team_bonus "2"

// Weapon weighting for DODS Stats
// Making a weight 2.0 will double the points given/taken
// Making a weight 0.5 will halve the points given/taken
mani_stats_dods_weapon_amerknife "3.0"
mani_stats_dods_weapon_spade "3.0"
mani_stats_dods_weapon_colt "1.6"
mani_stats_dods_weapon_p38 "1.5"
mani_stats_dods_weapon_c96 "1.5"
mani_stats_dods_weapon_garand "1.3"
mani_stats_dods_weapon_m1carbine "1.2"
mani_stats_dods_weapon_k98 "1.3"
mani_stats_dods_weapon_spring "1.5"
mani_stats_dods_weapon_k98_scoped "1.5"
mani_stats_dods_weapon_thompson "1.25"
mani_stats_dods_weapon_mp40 "1.25"
mani_stats_dods_weapon_mp44 "1.35"
mani_stats_dods_weapon_bar "1.2"
mani_stats_dods_weapon_30cal "1.25"
mani_stats_dods_weapon_mg42 "1.2"
mani_stats_dods_weapon_bazooka "2.25"
mani_stats_dods_weapon_pschreck "2.25"
mani_stats_dods_weapon_frag_us "1.0"
mani_stats_dods_weapon_frag_ger "1.0"
mani_stats_dods_weapon_smoke_us "5.0"
mani_stats_dods_weapon_smoke_ger "5.0"
mani_stats_dods_weapon_riflegren_us "1.3"
mani_stats_dods_weapon_riflegren_ger "1.3"
mani_stats_dods_weapon_punch "3.0"

// Bonus Points for DODS
mani_stats_dods_capture_point 4
mani_stats_dods_block_capture 4
mani_stats_dods_round_win_bonus 4

// ***************************************************************************
// Module : Victim Stats
//
// Desc : The victim stats module shows your statistic for the period when
//        you were alive
// ***************************************************************************

// Allow the use of victim stats
// 0 = off
// 1 = on
mani_show_victim_stats 0

// Set to 1 if you don't want to see damage taken from yourself
mani_show_victim_stats_inflicted_only 0

// This controls the default mode a player will have their victim stats mode
// set to when they first ever join your server. This setting is applied to
// the player's stored record withing player_settings.dat it does not control
```

```
// whether the victim stats functionality is on or off.
// 0 = mode 0, 1 = mode 1, 2 = mode 2, 3 = GUI mode
mani_player_settings_damage 0

// ****************************************************************************
// Module : Most Destructive
//
// Desc : This small module for CSS only shows the player that caused
//        the most damage with their kills at the end of a round
// ****************************************************************************

// Enable most destructive stats output
mani_stats_most_destructive 1

// 0 = By Kills then damage, 1 = by damage alone
mani_stats_most_destructive_mode 0

// This controls the default mode a player will have their *Most Destructve*
// stats mode set to when they first ever join your server. This setting is
// applied to the player's stored record withing player_settings.txt it
// does not control whether the *Most Destructve* functionality is on or off.
// 0 = disabled, 1 = enabled
mani_player_settings_destructive 1

// ****************************************************************************
// Module : Team Kill/Wound Protection
//
// Desc : The stats module is a very simple ranking system based on 3 methods of
//        calculation (defined in mani_stats_calculate).
// ****************************************************************************

// Enable TK Protection 1 = on, 0 = off
mani_tk_protection 1

// Enable TK Punishment menu 1 = on, 0 = off
mani_tk_forgive 1

// Time allowed after freezetime where spawn protection is enabled
mani_tk_spawn_time 5

// Defines whether bots can run tk punish options on other player (1 = on)
mani_tk_allow_bots_to_punish 0

// Defines whether when tk'ing a bot adds to a players tk violation count
// 0 = off, 1 = on
mani_tk_allow_bots_to_add_violations 0

// Number of tk violations before player is banned
mani_tk_offences_for_ban 7

// Time in minutes for a tk ban, 0 = permanent
mani_tk_ban_time 5

// When set to 1 a player will be slapped and have their view moved when
// team wounding.
mani_tk_slap_on_team_wound 0

// Sets the amount of damage a team wound inflicts on the attacker
mani_tk_slap_on_team_wound_damage 0

// If set to 1 shows opposition team wounds in chat and all team wounds if
// you are spectator, 0 = normal css style
mani_tk_show_opposite_team_wound 1

// Defines whether a players tk violation count is incremented even if
// forgiven (0 = off, 1 = on)
mani_tk_add_violation_without_forgive 0

// Turn on forgive option for tk punishments
mani_tk_allow_forgive_option 1

// Defines whether the blind option can be used on players (1 = on)
mani_tk_allow_blind_option 1
```

```
// Amount of blindness for 'blind' punishment (255 = completely blind)
mani_tk_blind_amount 253

// Turn on slap option for tk punishments
mani_tk_allow_slap_option 1

// Turn on tk cash option for tk punishments
mani_tk_allow_cash_option 1

// Amount of health that a player will slapped to
mani_tk_slap_to_damage 10

// Amount of cash to take from a team killer
mani_tk_cash_percent 30

// Turn on freeze option for tk punishments
mani_tk_allow_freeze_option 1

// When set to 1 the Drug option is allowed in the TK Menu
mani_tk_allow_drugged_option 1

// Turn on burn option for tk punishments
mani_tk_allow_burn_option 1

// This defines how long the burn time should be for in seconds for a
// tk punishment
mani_tk_burn_time 100

// Turn on slay option for tk punishments
// 1 = on, 0 = off
mani_tk_allow_slay_option 1

// Main option to turn on damage reflection
// 1 = on, 0 = off
mani_tk_team_wound_reflect 0

// Set the number of team wounds required by a player during the course
// of a map before the reflective damage kicks in
mani_tk_team_wound_reflect_threshold 5

// This value is a damage multiplier that inflicts whatever damage
// (armor loss + health loss) was given to the victim back to the attacker.
// When set to 1.0 the damage is perfectly reflected, set at 2.0
// the damage inflicted back on the attacker will be twice the damage etc.
mani_tk_team_wound_reflect_ratio 1.0

// An addition cvar here increases the reflection ratio each time the player
// teamwounds another player. This is to deter persistent team wounders
mani_tk_team_wound_reflect_ratio_increase 0.1

// This defines whether the tk time bomb option can be used or not
mani_tk_allow_time_bomb_option 1

// This defines the time before the bomb goes off
mani_tk_time_bomb_seconds 10

// This defines the bomb blast radius
mani_tk_time_bomb_blast_radius 1000

// 0 = no beams, 1 = beams on
mani_tk_time_bomb_show_beams 1

// 0 = player only, 1 = players on same team, 2 = all players
mani_tk_time_bomb_blast_mode 2

// This defines whether the tk fire bomb option can be used or not
mani_tk_allow_fire_bomb_option 1

// This defines the time before the bomb goes off
mani_tk_fire_bomb_seconds 10

// This defines the bomb blast radius
```

```
mani_tk_fire_bomb_blast_radius 1000

// 0 = no beams, 1 = beams on
mani_tk_fire_bomb_show_beams 1

// 0 = player only, 1 = players on same team, 2 = all players
mani_tk_fire_bomb_blast_mode 2

// Time in seconds that players will burn for
mani_tk_fire_bomb_burn_time 100

// This defines whether the tk freeze bomb option can be used or not
mani_tk_allow_freeze_bomb_option 1

// This defines the time before the bomb goes off
mani_tk_freeze_bomb_seconds 10

// This defines the bomb blast radius
mani_tk_freeze_bomb_blast_radius 1000

// 0 = no beams, 1 = beams on
mani_tk_freeze_bomb_show_beams 1

// 0 = player only, 1 = players on same team, 2 = all players
mani_tk_freeze_bomb_blast_mode 2

// Radius of beep circle, 0 = radius measures the same as the bomb blast,
// default is 256
mani_tk_time_bomb_beep_radius 0

// Radius of beep circle, 0 = radius measures the same as the bomb blast,
// default is 256
mani_tk_fire_bomb_beep_radius 0

// Radius of beep circle, 0 = radius measures the same as the bomb blast,
// default is 256
mani_tk_freeze_bomb_beep_radius 0

// Allow TK Beacon circle tk option
mani_tk_allow_beacon_option 1

// Radius of beacon circle
mani_tk_beacon_radius 384


// ****************************************************************************
// Module : Reserve Slot
//
// Desc : The Reserve slot module configuration cvars
// ****************************************************************************

// Turn on off reserve slots
mani_reserve_slots 0

// Number of reserve slots you have
mani_reserve_slots_number_of_slots 1

// User defined message shown in players console when kicked
mani_reserve_slots_kick_message "You were disconnected for using a reserve slot"

// User defined message for redirection of players to another server
mani_reserve_slots_redirect_message "This server is full, you are being redirected"

// The IP address of the server you wish to redirect players to. Leave it blank
// if you do not want redirection to be used
mani_reserve_slots_redirect ""

// This defines whether you want your reserve slots to fill with reserve players
// or always be kept free (1 = allow slots to fill, 0 = always keeps slots free
// and kick player instead)
mani_reserve_slots_allow_slot_fill 1

// Type of method used to kick players, 0 = by highest ping (spectators first),
```

```
// 1 = by connection time (spectators go first)
mani_reserve_slots_kick_method 1

// Include admins in the adminlist.txt file as players who have reserve slots
// (1 = include admins)
mani_reserve_slots_include_admin 1


// ****************************************************************************
// Module : High Ping kick
//
// Desc : The Reserve slot module configuration cvars
// ****************************************************************************

// Enable disable high ping kicker (1 = on)
mani_high_ping_kick 0

// Set the ping at which you want players kicked
mani_high_ping_kick_ping_limit 400

// Number of samples and averaged before a decision is made to kick a player
// (1 sample is about 1.5 seconds)
mani_high_ping_kick_samples_required 60

// Message displayed in console when player is disconnected
mani_high_ping_kick_message "Your ping is too high"


// ****************************************************************************
// Module : Admin action messages
//
// Desc : This section defines which admin actions will be anonymous to
//        non-admins on your server. If the option is set to 1 then the action
//        will not be shown to non-admins, if set to 0 then all players will
//        see which admin kicked/slayed/banned/etc a player.
// ****************************************************************************

mani_adminslap_anonymous 0
mani_adminblind_anonymous 0
mani_adminfreeze_anonymous 0
mani_admin teleport_anonymous 0
mani_admindrug_anonymous 0
mani_adminmap_anonymous 0
mani_adminswap_anonymous 0
mani_adminvote_anonymous 0
mani_adminsay_anonymous 0
mani_adminkick_anonymous 0
mani_adminslay_anonymous 0
mani_adminban_anonymous 0
mani_adminburn_anonymous 0
mani_adminnoclip_anonymous 0
mani_adminmute_anonymous 0
mani_admincash_anonymous 0
mani_adminsetskin_anonymous 0
mani_admindropc4_anonymous 0
mani_admintimebomb_anonymous 0
mani_adminfirebomb_anonymous 0
mani_adminfreezebomb_anonymous 0
mani_adminhealth_anonymous 0
mani_adminbeacon_anonymous 0
mani_admingravity_anonymous 0


// ****************************************************************************
// Module : Chat flooding control
//
// Desc : This is to control chat spam with the game
// ****************************************************************************

// Sets the time threshold for chat spamming (0 = off, 1.5 is a good
// value for use)
mani_chat_flood_time 0
```

```
// Sets the message the player will receive when they are spamming
mani_chat_flood_message "STOP SPAMMING THE SERVER !!"


// ****************************************************************************
// Module : Basic auto balance teams
//
// Desc : This is a very basic auto balancer, this does not take into
//        account player skill
// ****************************************************************************

// Set to 1 for auto team balancing at end of rounds (an alternative to the
// CSSource built in team balancer)
mani_autobalance_teams 0

// 0 = Players balanced regardless if dead or alive, 1 = dead players swapped
// first followed by alive players, 2 = only dead players can be swapped
mani_autobalance_mode 1


// ****************************************************************************
// Module : Current Time Display
//
// Desc : This controls the time of day message formatting
// ****************************************************************************

// 1 for military style time, 0 for 12 hour clock
mani_military_time 1

// Set to your servers timezone or leave it blank, it will be added to the
// end of the time when displayed
mani_thetime_timezone "GMT"

// mani_adjust_time cvar allows you to specify the number of minutes to
// add or subtract from your server clock when you type thetime in game.
// If you server is 20 minutes fast set mani_adjust_time -20, if it is
// 30 minutes slow set mani_adjust_time 30.
mani_adjust_time 0


// ****************************************************************************
// Module : Voting functionality
//
// Desc : There are two types of vote. System started where an admin has
//        triggered a vote or User started where a user has started a vote
//        The following cvars control the configuration of the voting
// ****************************************************************************

// Allow voting 1 = on, 0 = off (this cvar controls ALL voting)
mani_voting 1

// Defines the last number of maps played to not show in random votemap lists
mani_vote_dont_show_last_maps 3

// Defines the time in minutes a extend vote will add to the timeleft counter
mani_vote_extend_time 20

// Defines the whether the a map can be extended
mani_vote_allow_extend 1

// Defines amount of time in seconds a vote will be allowed for
mani_vote_allowed_voting_time 45

// Defines whether a random map vote will be displayed towards the end of
// the map
mani_vote_allow_end_of_map_vote 0

// Number of extensions a map is allowed via user vote or random map vote,
// 0 = infinite
mani_vote_max_extends 2

// Number of rounds to extend by if mp_winlimit is not 0
mani_vote_extend_rounds 10
```

```
// Define the file to use for random map vote
// 0 = mapcycle.txt, 1 = votemapslist.txt, 2 = maplist.txt
mani_vote_mapcycle_mode_for_random_map_vote 2

// Define the file that admin can select from for admin
// started vote.
// 0 = mapcycle.txt, 1 = votemapslist.txt, 2 = maplist.txt
mani_vote_mapcycle_mode_for_admin_map_vote 2

// Defines how many minutes before the end of the map that a random map vote
// is started
mani_vote_time_before_end_of_map_vote 3

// Defines how many maps can be in the end of map vote
mani_vote_max_maps_for_end_of_map_vote 6

// Allow team swap option as part of Extend map on end
// of map vote (CSS Only)
mani_vote_end_of_map_swap_team 0

// Defines the vote percentage required to set map
mani_vote_end_of_map_percent_required 60

// Defines the vote percentage required to set rcon vote
mani_vote_rcon_percent_required 60

// Defines the vote percentage required to set question vote
mani_vote_question_percent_required 60

// Defines the vote percentage required to set map vote
mani_vote_map_percent_required 60

// Defines the vote percentage required to set random map vote
mani_vote_random_map_percent_required 60

// This cvar determines how the players see the votes during voting
// 0 = quiet mode,
// 1 = show players as they vote but not their choice,
// 2 = Show voted choice but not player,
// 3 = show player name and their choice
mani_vote_show_vote_mode 3

// Following cvar now has 2 modes of operation
// 0 = alive players will see vote menu,
// 1 = alive players will need to type vote to access the menu,
mani_vote_dont_show_if_alive 0

// Allow user started votemaps
mani_vote_allow_user_vote_map 1

// Allow the users to extend maps if time based
mani_vote_allow_user_vote_map_extend 1

// Allow the users to kick players by vote
mani_vote_allow_user_vote_kick 1

// Allow the users to ban players by vote
mani_vote_allow_user_vote_ban 0

// Defines the vote percentage required to set an extend map vote
mani_vote_extend_percent_required 60

// Percentage of votes required from players before map change
mani_vote_user_vote_map_percentage 60

// Time after a new map starts that voting is allowed
mani_vote_user_vote_map_time_before_vote 60

// Minimum number of votes required to change a map (override vote percentage)
mani_vote_user_vote_map_minimum_votes 4

// 0 = only when no admin on server, 1 = all the time
```

```
mani_vote_user_vote_kick_mode 0

// Percentage of votes required from players before kick occurs
mani_vote_user_vote_kick_percentage 60

// Time after a new map starts that voting is allowed
mani_vote_user_vote_kick_time_before_vote 60

// Minimum number of votes required (override vote percentage)
mani_vote_user_vote_kick_minimum_votes 4

// 0 = only when no admin on server, 1 = all the time
mani_vote_user_vote_ban_mode 0

// Percentage of votes required from players before kick occurs
mani_vote_user_vote_ban_percentage 60

// Time after a new map starts that voting is allowed
mani_vote_user_vote_ban_time_before_vote 60

// Minimum number of votes required (override vote percentage)
mani_vote_user_vote_ban_minimum_votes 4

// Time in minutes for the ban, 0 = permanent ban
mani_vote_user_vote_ban_time 30

// 0 = ban by ID, 1 = ban by IP, 2 = ban by ID and IP
mani_vote_user_vote_ban_type 0

// Allow rock the vote
mani_vote_allow_rock_the_vote 1

// Defines the vote percentage required to set map
mani_vote_rock_the_vote_percent_required 60

// Time before rockthevote can be started after a new map starts
mani_vote_time_before_rock_the_vote 120

// Number of nominations included in the vote
mani_vote_rock_the_vote_number_of_nominations 4

// Number of random maps chosen from votemaplist.txt
mani_vote_rock_the_vote_number_of_maps 8

// Percentage of players on server required to type rockthevote before
// it starts
mani_vote_rock_the_vote_threshold_percent 60

// Minimum number of players required to type rockthevote before it starts
mani_vote_rock_the_vote_threshold_minimum 4

// This controls the default mode a player will have their 'show vote progress'
// set to when they first ever join your server. This setting is applied to
// the player's stored record withing player_settings.dat it does not control
// whether the vote progress functionality is on or off.
// 0 = player settings default to off, 1 = player settings default to on
// 1 = default on, 0 = default off
mani_player_settings_vote_progress 1

// ****************************************************************************
// Module : Word filter module
//
// Desc : The following cvars control the configuration of the chat word filter
// ****************************************************************************

// 0 = off, 1 = show warning to player
mani_filter_words_mode 0

// Message shown to player
mani_filter_words_warning "SWEARING IS NOT ALLOWED ON THIS SERVER !!!"


// ****************************************************************************
```

```
// Module : Sounds Control
//
// Desc : The following cvars control how system sounds are configured. This
//        is not related to the Quake sounds
// ********************************************************************************

// Set to the number of sounds you wish a regular non-admin player to be able
// to use per round
mani_sounds_per_round 0

// Set to 1 if you want alive players not to hear sounds triggered by dead
// players
mani_sounds_filter_if_dead 0

// mani_sounds_auto_download is a cvar to control whether server sounds
// (not quake sounds) are auto downloaded to a client. If set to 0 you must
// provide your own .res files to initiate transfers to a client.
// If you change this value from 1 to 0 while the server is running you
// must restart your server.
mani_sounds_auto_download 1

// This controls the default mode a player will have their death beam mode
// set to when they first ever join your server. This setting is applied to
// the player's stored record withing player_settings.dat it does not control
// whether the death beam functionality is on or off.
// 0 = player settings default to off, 1 = player settings default to on
// 1 = default on, 0 = default off
mani_player_settings_sounds 1

// ********************************************************************************
// Module : Plugin Logging
//
// Desc : The following cvars control how the plugin logging is configured
// ********************************************************************************

// Admin logging parameters
// Directory where logs will be stored under the mani_path directory
// The logging mode you wish to use
// 0 = default placing of log files in the same .log files that Valve creates
// 1 = logs created per map change using the same style filenames that Valve
// uses in the mani_log_directory directory
// 2 = One large file is written in the mani_log_directory
// 3 = A log is written as a steam id for each admin that runs a command, the
// format is STEAM_x_x_xxxxxxxx.log
mani_log_mode 0

// Path where the logs are stored
mani_log_directory "mani_logs"


// ********************************************************************************
// Module : Death Beams
//
// Desc : The death beams show a solid beam when you die between the point
//        where your attacker killed you from and the point where you
//        were when you died. Only the victim will see the beam
// ********************************************************************************

// 0 = death beams not allowed, 1 = death beams are shown if player
// has them turned in their settings
mani_show_death_beams 0

// This controls the default mode a player will have their death beam mode
// set to when they first ever join your server. This setting is applied to
// the player's stored record withing player_settings.dat it does not control
// whether the death beam functionality is on or off.
// 0 = player settings default to off, 1 = player settings default to on
mani_player_settings_death_beam 1


// ********************************************************************************
// Module : Anti IP Ghosting
//
```

```
// Desc : The plugin can be configured to blind players on the same IP once
//        they die and another player is still alive. At the point when all
//        players ghosting on the same IP are dead they will be able to see
//        again.
// ************************************************************************

// This cvar prevents players on the same IP from viewing the game whilst
// another player on the same IP is still alive. Admins on the same IP are
// immune from this and there is an immunity flag that can also be used.
// 0 = Dont blind ghosters on same IP, 1 = blind ghosters on same IP address
mani_blind_ghosters 0

// The following cvars can prevent players using the same IP taking over
// your server by having a majority vote and kicking players so they can play.
// 0 = Players on same IPs cannot use voteban
// 1 = Players on same IPs can use voteban
mani_vote_allow_user_vote_ban_ghost 1

// 0 = Players on same IPs cannot use votekick
// 1 = Players on same IPs can use votekick
mani_vote_allow_user_vote_kick_ghost 1


// ************************************************************************
// Module : Decal Map Adverts
//
// Desc : In game map adverts can be placed onto static objects in the game
//        such as walls.
// ************************************************************************

// 1 = turn on map adverts, 0 = turn off map adverts
mani_map_adverts 0

// Note about the following cvar. If you enable war mode with
// mani_map_adverts_in_war 0 then you must either change map or get the
// players to type 'retry' in their console for the existing adverts to disappear
// 1 = allow map adverts with war mode on
// 0 = disallow map adverts in war mode
mani_map_adverts_in_war 0


// ************************************************************************
// Module : Anti-cheat
//
// Desc : For the most part this code is now out of date as Vac should pick
//        up the cheat type that this looks for so it is not worth turning on
//        the detection at the current time.
// ************************************************************************

// 0 = off otherwise set to the number of name changes allowed before
// action is taken (this was for a hack that constantly changed a player's
// name making it hard to find the actual player to ban)
mani_player_name_change_threshold 15

// 0 = reset name change count per round
// 1 = reset name change count per map
mani_player_name_change_reset 0

// 0 = kick, 1 = ban by ID, 2 = ban by IP, 3 = ban by ID and IP
mani_player_name_change_punishment 0

// 0 = permanent ban otherwise specifies the number of minutes
mani_player_name_change_ban_time 0

// ************************************************************************
// Module : Extra spawnpoints
//
// Desc : Extra spawnpoints can be turned on and off using the following cvar
// ************************************************************************

// 0 = off, 1 = on (will use spawnpoints.txt file)
mani_spawnpoints_mode 0
```

```
// ************************************************************************
// Module : Custom Skin Control
//
// Desc : The following cvars are used to configure the options for the skins
// ************************************************************************

// 0 = Dont allow admins to have admin skins, 1 = Allow admins to have admin
// skins
mani_skins_admin 0

// 0 = Dont allow public skins for normal players, 1 = Allow public skins
// for normal players
mani_skins_public 0

// 0 = Dont force first skin in list for public player, 1 = Force first skin
// in list on public player
mani_skins_force_public 0

// 0 = Allow all skins to be selected via ma_setskin, 1 = Only allow misc
// skins to be used
mani_skins_setskin_misc_only 0

// 0 = Dont auto download skin resources, 1 = auto download skin resources
// to clients
mani_skins_auto_download 0

// 0 = Dont allow immunity players to have reserved skins,
// 1 = Allow immunity players to have reserved skins
mani_skins_reserved 0

// 0 = No menu on team join, 1 = show skin chooser on team join,
// 2 = show settings menu on team join
mani_skins_force_choose_on_join 1

// 0 = no custom skins for bots, 1 = use random public class skins on bots
mani_skins_random_bot_skins 1


// ************************************************************************
// Module : Spray Tag Tracking
//
// Desc : The spray tag tracking module allows an admin to check the details
//        of who sprayed a tag within the game using the command ma_spray
//        while near the spray in question
// ************************************************************************

// Spray Tag Tracking control
// 0 = off, 1 = on
mani_spray_tag 0

// Time in seconds that a spray will be tracked for
mani_spray_tag_spray_duration 120

// Max distance away from a spray that you can be for acquisition
mani_spray_tag_spray_distance_limit 500

// Use effect to show which spray is being targetted
// 0 = none
// 1 = beam (defaults to glow for DoD)
// 2 = glow
mani_spray_tag_spray_highlight 1

// Non-permanent ban time in minutes
mani_spray_tag_ban_time 60

// Console messages
mani_spray_tag_warning_message "Please stop using your spray"
mani_spray_tag_kick_message "You have been kicked for using an offensive spray"
mani_spray_tag_ban_message "You have been banned for 60 minutes through using an offensive spray"
mani_spray_tag_perm_ban_message "You have been permanently banned for using an offensive spray"

// 0 = Allow sprays on the server
// 1 = Block all sprays on the server (must have mani_spray_tag 1)
```

```
mani_spray_tag_block_mode 0

// Warning message if sprays are blocked
mani_spray_tag_block_message "Sprays are blocked on this server !!"

// Amount of damage to inflict for a spray tag warn with slap option
mani_spray_tag_slap_damage 0


// ****************************************************************************
// Module : Warmup Timer
//
// Desc : Warmup timers are used at the start of a map. A timer can be setup
//        that will restart the map after a set amount of time. For something
//        like CSS this means late joiners can still start on the pistol round
//        and not miss out. For CSS there is a specific cvar to allow only
//        knives to be used during this period.
// ****************************************************************************

// 0 = No warmup time on map load
// Greater than 0 = number of seconds after map load until map restarts and
// play continues as normal.
mani_warmup_timer 0

// 1 = visible countdown displayed in center of screen
// 0 = no countdown displayed
mani_warmup_timer_show_countdown 1

// For CSS only, set to 1 for knife mode where players are only allowed to use
// knives during the warmup period
mani_warmup_timer_knives_only 0

// For CSS only, allow players to respawn when dying in the warmup knife round
// only
mani_warmup_timer_knives_respawn 1

// Set the following to 1 if you do not want TK punishments to be used during
// the warmup round. Set to 0 if you wish to allow tk punishments
mani_warmup_timer_ignore_tk 1

// If you want the knife mode to be disabled on fy_ and aim_ maps where guns
// are already on the ground set this to 1, set to 0 if you do not care
mani_warmup_timer_knives_only_ignore_fyi_aim_maps 1

// Set this to 1 for unlimited HE Grenades during warmup (this is really good fun !!)
mani_warmup_timer_unlimited_grenades 0

// Items to give to player at spawn i.e weapon_ak47, item_assaultsuit etc
// You can have multiple items on each cvar, the plugin will pick a random item
// from each cvar for example
// mani_warmup_timer_spawn_item_2 "weapon_xm1014:weapon_usp:weapon_ump45:weapon_tmp:weapon_scout:wea
pon_p90:weapon_aug:weapon_p228:weapon_mp5navy:weapon_mac10:weapon_m4a1:weapon_m3:weapon_m249:wea
pon_glock:weapon_galil:weapon_fiveseven:weapon_ak47"

mani_warmup_timer_spawn_item_1 "item_assaultsuit"
mani_warmup_timer_spawn_item_2 ""
mani_warmup_timer_spawn_item_3 ""
mani_warmup_timer_spawn_item_4 ""
mani_warmup_timer_spawn_item_5 ""

// If friendly fire is enabled normally setting this to 1 will disable it
// during the course of the warmup
mani_warmup_timer_disable_ff 1

// Set infinite ammo for CSS, 0 = disabled, 1 = enabled
mani_warmup_infinite_ammo 0

// ****************************************************************************
// Module : Menu options
//
// Desc : The following options determine the type of menu you wish to use
//        and how they are displayed
// ****************************************************************************
```

```
// Set to 1 for in game amx style menus, or 0 for Escape style menus
// This is overriden by the gametypes.txt file if the game does not support
// AMX style menus. At the time of writing there are only 3 games that support
// the AMX style. CS:S, DoD:S and HL2CTF
mani_use_amx_style_menu 1

// 0 = No sorting of menus, 1 = Sort most menus by players name
mani_sort_menus 1


// ****************************************************************************
// Module : External logging (V1.2 required)
//
// Desc : This configures extra logging that is required by extern stats
//        programs like hlstatsx and psychostats for body hit group counts
//        and accuracy related to weapon types.
// ****************************************************************************

// Enables disables the logging 0 = off, 1 = on
mani_external_stats_log 0

// Option to allow extra logs within war mode, 0 = off, 1 = on
mani_external_stats_log_allow_war_logs 0

// Option for CSS to log bot kills or not
// 0 = do not log bot kills
// 1 = log bot kills
mani_external_stats_css_include_bots 0


// ****************************************************************************
// Module : Save scores
//
// Desc : This module saves player scores when they leave a map then restores
//        their scores if they then rejoin on the same map. For CSS this can
//        also include cash if the option is set. If a map is restarted via
//        mp_restartgame the saved scores are reset
// ****************************************************************************

// Enabled/Disabled save scores functionality, 0 = off/1 = on
mani_save_scores 0

// Amount of time in minutes to store a player score for, if set to 0 the score
// will be tracked for the duration of the map
mani_save_scores_tracking_time 5

// For CSS only if set to 1 the save scores module will also restore their cash
// too. 0 = off/1 = on
mani_save_scores_css_cash 1


// ****************************************************************************
// Module : Auto Join restriction
//
// Desc : This module can stop a player from selecting a specific team and
//        force them to join using auto-assign or join spectator. As an
//        extension to this it can also remember the team that the player is
//        assigned to and force them to rejoin that team if they re-connect on
//        that map. The auto-join module can be overriden by ma_swapteam,
//        auto balancing and any extern plugins such as etb or ptb.
// ****************************************************************************

// Option to enabled/disable the functionality. By having it enabled the player
// must use auto-assign when joining a team
// 1 = on/0 = off
mani_team_join_force_auto 0

// When set to 1 the plugin will store the team the player was on and force them
// to that team if they try to re-connect or change team within the game.
mani_team_join_keep_same_team 0
```

```
// ****************************************************************************
// Module : Steam ID Pending kicker
//
// Desc : This module can kick a player whos steam ID is stuck at
//        STEAM_ID_PENDING after a certain time period. When kicked the user
//        will be shown a message asking them to re-connect
// ****************************************************************************

// Option to specify the number of seconds to wait after the player joins the
// server before kicking them. A suggested value is about 15-20 seconds.
// If set to 0 the functionality is disabled
mani_steam_id_pending_timeout 0

// An option to display to admin when a player was kicked for having a
// steam id pending problem.
mani_steam_id_pending_show_admin 1


// ****************************************************************************
// Module : AFK Manager
//
// Desc : This module can control how to manage players who are AFK. It works
//        by tracking keyboard and mouse movements rather than in game player
//        positions. The manager can be configured to work using the number
//        of rounds that a player is AFK for or by a set number of seconds
//        a player can be AFK.
// ****************************************************************************

// Cvar to turn on/off the AFK Manager 0 = off, 1 = on
mani_afk_kicker 0

// 0 = Kick to spectator first, then off the server, 1 = Kick straight off the
// server
mani_afk_kicker_mode 0

// 0 = disabled, > 0 = number of rounds before a player is kicked off server or
// to spectator
mani_afk_kicker_alive_rounds 0

// 0 = disabled, > 0 = number of rounds before being kicked off server
mani_afk_kicker_spectator_rounds 0

// 0 = disabled, > 0 = number of seconds before a player is kicked off server or
// to spectator
mani_afk_kicker_alive_timer 0

// 0 = disabled, > 0 = number of seconds before being kicked off server
mani_afk_kicker_spectator_timer 0

// 0 = Immune player is not moved at all, 1 = Immune player will be moved to spectator
// but not kicked
mani_afk_kicker_immunity_to_spec_only 0

// ****************************************************************************
// Module : Betting Module
//
// Desc : This module is a simple betting module similar to the Mattie's
//        event script by ajax though not as complex.
// ****************************************************************************

// 0 = Disable betting, 1 = Enable betting
mani_css_betting 0

// 0 = Players can bet when alive or dead, 1 = Players can only bet when dead
mani_css_betting_dead_only 1

// This determines if a in a x vs 1 situation if the single player who the
// odds are against wins, they receive the losing wager pot.
// The setting determines at what number of players that this option applies
// e.g 5 vs 1, you would set it to 5, 3 vs 1, set it to 3.
// In the example of 5 vs 1. If the total bets made total $4000 for the team of
// 5 and they lose, the player who killed all 5 will receive the $4000 that was
// wagered against that player.
```

```
mani_css_betting_pay_losing_bets 0

// 0 = No announcement, 1 = announcement made to place bets
mani_css_betting_announce_one_v_one 0


// ****************************************************************************
// Module : Bounty Module
//
// Desc : This module is a bounty module similar to that of Firesnakes Script
//        for Matties event scripts. The bounty module tracks a players kill
//        streak. Once past a certain amount of kills that player has a bounty
//        placed on their head. The person who then kills that player receives
//        the bounty
// ****************************************************************************

// 0 = Disable, 1 = Enable the module
mani_css_bounty 0

// Sets the kill streak required to have a bounty placed on a player
mani_css_bounty_kill_streak 5

// Sets the start bounty for the player
mani_css_bounty_start_cash 1000

// Sets the amount of cash add to a players bounty for surviving a round with
// a bounty on their head
mani_css_bounty_survive_round_cash 500

// Sets the amount of cash added to a players bounty for killing another player
// whilst having a bounty on themseleves
mani_css_bounty_kill_cash 250

// Sets the colour a player should turn into when a bounty is on them when playing
// as CT
mani_css_bounty_ct_red 255
mani_css_bounty_ct_green 255
mani_css_bounty_ct_blue 255
mani_css_bounty_ct_alpha 255

// Sets the colour a player should turn into when a bounty is on them when playing
// as T
mani_css_bounty_t_red 255
mani_css_bounty_t_green 255
mani_css_bounty_t_blue 255
mani_css_bounty_t_alpha 255


// ****************************************************************************
// Module : Objectives Module for CSS
//
// Desc : This module is an objectives module that slays all players who have
//        not completed their team objectives at the end of a round. For example
//        if the CTs choose not to defuse a bomb, they will be slayed. If
//        the terrorists refuse to plant the bomb, they will be slayed at the
//        end of the round.
// ****************************************************************************

// 0 = Disable, 1 = Enable
mani_css_objectives 0

// ****************************************************************************
// Module : AutoMap Module
//
// Desc : This module is designed to allow a set of popular maps to be
//        autoloaded if your server has a minimum number of players on it.
//        A typical setup would be to switch to de_dust2 for instance if there
//        are no players on the server at the time.
// ****************************************************************************

// 0 = Disable, 1 = Enable
mani_automap 0
```

```
// This cvar sets up the list of maps that you would like to switch to, you
// can setup more than one map by seperating each map with the : symbol e.g
// mani_automap_map_list "de_dust2:de_aztec:cs_office"
mani_automap_map_list ""

// The number of players or less that will trigger the automap change
mani_automap_player_threshold 0

// Include bots in the threshold calculation.
// 0 = Exclude bots from the calculation
// 1 = Include bots in the calculation
mani_automap_include_bots 0

// Time in seconds after the player threshold has been reached that an automap
// change will take place. Default is 5 minutes (300 seconds)
mani_automap_timer 300

// If an automap event happens, you can set the next map once that map has
// loaded to be the same as the current map.
// 0 = disabled
// 1 = enabled
mani_automap_set_nextmap 0


// ****************************************************************************
// Module : Miscallaneous
//
// Desc : Small cvars for various controls
// ****************************************************************************

// Call to the mani_quake_sounds.cfg file if it exists
exec mani_quake_sounds.cfg

// The following cvar controls how the mapcycle is calculated
// 0 = standard Valve map cycle,
// 1 = if you don't want your mapcycle to reset to the first in the
// list when moving to a map not in the cycle,
// 2 = random cycle (uses mani_vote_dont_show_last_maps cvar to exclude
// last maps played)
// 3 = skip to the next unplayed map in the map cycle list until all maps
// have been played when it is reset.
mani_mapcycle_mode 0

// 0 = Normal game play
// 1 = players get free HEs in CSS at spawn and after an HE has been thrown
mani_unlimited_grenades 0

// Force all dead players to run overview_mode 0 every game frame
mani_war_mode_force_overview_zero 0

// Set this to allow stacking of players in CSS
// (overrides cs_stacking_num_levels cvar)
mani_cs_stacking_num_levels 2

// mani_use_ma_in_say_command When set to 1 you must use the prefix ma_ in
// say commands, if 0 you can drop the ma_ prefix if you wish
// This is only for in game say commands and for Beetlefart compatibilty if
// you use it.
mani_use_ma_in_say_command 0

// 0 = All Dead talk off, 1 = All dead talk on
mani_dead_alltalk 0

// Set to 0 for normal mode, set to 1 for spam removal.
// This cvar is useful if you are using Mattie's Event Scripts that call
// plugin functions so the chat area is not spammed.
mani_mute_con_command_spam 0

// Enable disable top left ma_say and ma_chat
mani_adminsay_top_left 1

// Enable disable chat area ma_say and ma_chat
mani_adminsay_chat_area 1
```

```
// Allow admin say at bottom of screen
mani_adminsay_bottom_area 0

// Allow users to chat to admins using ma_chat
mani_allow_chat_to_admin 1

// Defines whether the command ff executed is shown only to the player or the
// whole server (1 = player only)
mani_ff_player_only 0

// Defines whether the command nextmap executed is shown only to the player or
// the whole server (1 = player only)
mani_nextmap_player_only 0

// Defines whether the command timeleft executed is shown only to the player
// or the whole server (1 = player only)
mani_timeleft_player_only 0

// Defines whether the command thetime executed is shown only to the player or
// the whole server (1 = player only)
mani_thetime_player_only 0

// This defines how long the burn time should be for in seconds when a player
// is burned by admin
mani_admin_burn_time 20

// This defines whether a message will be displayed when a hostage stops
// following you in CSS.
mani_hostage_follow_warning 0

// This allows you to change the prefix of chat commands, the default is @
mani_say_command_prefix "@"

// This allows all players to be able to use the ma_rates command
// 0 = Only admins with ma_rates access,
// 1 = Anyone on the server can use ma_rates
mani_all_see_ma_rates 0

// If a team swap takes place via the end of map vote, set this to 1 to
// swap the team scores, set to 0 if not (CSS Only)
mani_swap_team_score 1

// If you want the menus to close automatically after making a select like
// slay, freeze etc, set this to 1. If you wish them to stay open set this to 0
mani_old_style_menu_behaviour 0

// If you are using the AMX style menus but would prefer the option to use
// the escape style text input box when inputting text data such as when
// setting a name in the Client Admin menu, then set this to 1.
mani_menu_force_text_input_via_esc 0

// With the following cvar you can setup the maximum time that a player can ban
// for if they only have the normal ban flag 'b' setup for them. The flag for
// any kind of ban incuding permanent bans is the 'pban' flag. The value of the
// cvar is in minutes. The default value is 360 minutes or 6 hours, any admin
// with the 'b' flag but not the 'pban' flag will only be able to ban a player
// for 6 hours at the most and will not be able to do permanent bans.
mani_admin_temp_ban_time_limit 360

// This cvar slays players who are killed during a CSS round then leave and
// rejoin the server in the same round then spawn to play the same round
// more than once. 0 = Disabled, 1 = Enabled
mani_anti_rejoin 1

// This cvar will refund the money for a weapon if it is removed at spawn
// from a player due to an impending restriction.
mani_weapon_restrict_refund_on_spawn 1

// This cvar will prevent users from picking up a weapon if it is restricted.
// It also closes a loop hole where a weapon can be bought, dropped, then
// another weapon bought (assuming 1 weapon per team is in effect)
mani_weapon_restrict_prevent_pickup 1
```

```
// 5 configs that can be run on start of map load
// mani_exec_default_file1 defaults if not set to mani_server.cfg
// mani_exec_default_file2 defaults if not set to ./mani_admin_plugin/defaults.cfg

mani_exec_default_file1 "mani_server.cfg"
mani_exec_default_file2 "./mani_admin_plugin/defaults.cfg"
mani_exec_default_file3 ""
mani_exec_default_file4 ""
mani_exec_default_file5 ""

// If you have Steam Bans running (www.steambans.com) you can automatically
// get your client to run sb_status in the console when choosing a player
// to observe. 0 = disabled, 1 = enabled.
mani_sb_observe_mode 0

// ****************************************************************************
// Finish up with a message to the console
// ****************************************************************************

echo "****** Finished executing mani_server.cfg ******"
```

A.4 server.cfg for ETH1

```
// ------------------------  Server Name (Hostname) ------------------

// hostname des Servers
hostname "ETH 1: DFSOCOM Battlegrounds"

//-------------------------regionaleinstelung-------------------------
// Must specify sv_region, or it won't show up in Steam server browser

// 0: US East coast
// 1: US West coast
// 2: South America
// 3: Europe
// 4: Asia
// 5: Australia
// 6: Middle East
// 7: Africa
// 255:

sv_region 3


//-------------------------- CS Specific Cvars ----------------------

//Dynamic Pricing (1=on)
mp_dynamicpricing 0


// Informationen ueber andere Spieler in der Statusbar
// 0=alle 1=team 2=keine
mp_playerid 0

// Autokick fuer "idle Clients" & Ban Teamkiller? 1=an 0=aus
mp_autokick 0

// Einschalten der automatischen Balance der Teams. 1=an 0=aus
mp_autoteambalance "1"

// Dauer in Sekunden bis C4 explodiert
// Range is 15 - 90. =sekunden
mp_c4timer 35

// Taschenlampe darf benutzt werden 1=an 0=aus
mp_flashlight "1"
```

```
// Footsteps an? 1=an 0=aus
mp_footsteps 1

// erzwinge Verfolgerkamera?
// 0=freie Sicht 1=nur eigenes Team 3=Ort an dem man stirbt bleibt fix
mp_forcechasecam 1

// schwarzer Bildschirm nachdem man stirbt? 1=an 0=aus
mp_fadetoblack 0

// 0 zum abschalten. kennzeichnet die Laenge der Freeze Periode bei jedem Start.
mp_freezetime 2

// Zeit, in der man noch nach Rundenbegin kaufen kann (float)
mp_buytime 0.5

//Geld das am Anfang einer neuen Map zur Verfuegung steht
mp_startmoney 3500

// Spectators erlauben? 1=an 0=aus
allow_spectators 1

// 1=Friendlyfire ist an! 0=aus
mp_friendlyfire 1

// lowlag Eintrag fuer Internet-Server, fuer LAN weglassen
mp_lowlag "1"

// Granaten verletzen eigenes Team 1=an 0=aus
mp_friendly_grenade_damage 1

 // Bei 2 Hostagekills vom Server kicken  0=aus
mp_hostagepenalty 4

// maximum von Spielern die ein Team mehr Ueberzaehlig sein kann. 0=aus
mp_limitteams 1

// Logge Chat Messages?
// verwende die Server Admins zum ausgeben der Chatnachrichten in den Log Daten.
mp_logmessages 1

// Dauer einer Runde in Minuten
// Minimum der Runde ist 1 Min. Maximum ist 9 Min. 1.5 ist 90 Sekunden
mp_roundtime 4

// Maximale Anzahl von Runden bis der naechste Mapwechsel erfolgt.
mp_maxrounds 200

// Maximale Dauer einer Map in Minuten
// (Zeit bis die naechste Map Automatisch geladen wird.)
mp_timelimit 40

// Laet die naechste Map wenn ein Team so viele Gewinne erzielt hat. 0=aus
mp_winlimit 0

// Teamkills werden mit Aussetzen bestraft 1=an 0=aus
mp_tkpunish "0"

//Prozentsatz der Spieler die eine Map waehlen muessen bevor dies geladen wird. 0.0 bis 1.0
mp_mapvoteratio 0.5

// Sets the amount of time in minutes before resetting the game,
// including frags, weapons, money, and scores -
// good for clan practice rounds. Set to '0' to disable.
sv_restartround 0

sv_enableoldqueries 1

//---------------------- General HL Cvars-----------------------

// Sets the amount of time in seconds in between client logo sprays.
decalfrequency 60
```

```
// Ob der Spieler schaden nimmt, falls er von groesseren Hoehen springt/faellt 0=aus 1=an
mp_falldamage 1

// Clients duerfen das Spiel nicht pausieren 0=aus 1=an
pausable 0

// automatisches Zielen deaktiviert  1=aktiviert
sv_aim 0

// Cheats deaktiviert 1=aktiviert
sv_cheats 0

// maximale Geschwindigkeit der Spieler
sv_maxspeed 320

// maximale Datenrate die an den Client gesendet wird
sv_maxrate 15000

// minimale Datenrate die an den Client gesendet wird
sv_minrate 4000

// Wie lange jemand in der Luft bleiben kann.
sv_airaccelerate 2

//---------------------------- Rcon & Password Stuff --------------

 // WICHTIG! rcon Passwort fuer Fernadministrierung
rcon_password "marinebiol"

// Evtl. vergebenes Passwort fuer den Server
sv_password ""

//----------------------------  Banned.cfg ---------------

// In the event that you have a llist of banned clients.
// this executes the banned.cfg file so that they will
// remain banned even after map changes.
exec banned.cfg

//------------------------- Cheating-Death -----------------------------

// Bei verwendung von Cheating-Death cdrequired 1,
//cdrequiredmsg,cdoptionalmsg,cdupdatemsg das // davor endfernen
// beziungsweise mit // auskommentiren bei nicht verwendung von Cheating-Death

// CD per rcon an,-ausmachen 0=optionat 1=an -1=aus
//cdrequired 1

// CD Req Nachricht:
//cdrequiredmsg "Cheating-Death ist Zwang!"

// CD optional Nachricht:
//cdoptionalmsg "Cheating-Death ist optional."

// CD Versions Update Nachricht:
//cdupdatemsg "Bitte dein Cheating-Death updaten!"

//---------------------------- server logging.----------------------

// Logfiles werden geschrieben ("+log on" in der Kommandozeile) 1=an 0=aus
mp_logfile 1
log on

//benutzt CVAR um den Level der anzeige einzustellen.
//( 0 = log enemy attacks. 1 = log Teammate attacks)
mp_logdetail 1


//bestimmt, ob ein logfile geschrieben wird, oder eins per map
sv_log_onefile 1
//---------------------------- Bot control---------------------

//sollen Bots sprechen?
```

```
bot_chatter off

//Bot skill level
bot_difficulty 2

//Duerfen Bots ohne Menschen spielen?
bot_join_after_player 0

//Wieviele Bots?
bot_quota 4

bot_quota_mode fill

bot_auto_vacate 1
```

A.5 server.cfg for ETH2

```
// -----------------------  Server Name (Hostname) -----------------

// hostname des Servers
hostname "ETH 2: DUST2 Only"

//-------------------------regionaleinstelung-------------------------
// Must specify sv_region, or it won't show up in Steam server browser

// 0: US East coast
// 1: US West coast
// 2: South America
// 3: Europe
// 4: Asia
// 5: Australia
// 6: Middle East
// 7: Africa
// 255:

sv_region 3


//-------------------------- CS Specific Cvars ----------------------

//Dynamic Pricing (1=on)
mp_dynamicpricing 0

// Informationen ueber andere Spieler in der Statusbar
// 0=alle 1=team 2=keine
mp_playerid 0

// Autokick fuer "idle Clients" & Ban Teamkiller? 1=an 0=aus
mp_autokick 0

// Einschalten der automatischen Balance der Teams. 1=an 0=aus
mp_autoteambalance "1"

// Dauer in Sekunden bis C4 explodiert
// Range is 15 - 90. =sekunden
mp_c4timer 35

// Taschenlampe darf benutzt werden 1=an 0=aus
mp_flashlight "1"

// Footsteps an? 1=an 0=aus
mp_footsteps 1

// erzwinge Verfolgerkamera?
// 0=freie Sicht 1=nur eigenes Team 3=Ort an dem man stirbt bleibt fix
```

```
mp_forcechasecam 1

// schwarzer Bildschirm nachdem man stirbt? 1=an 0=aus
mp_fadetoblack 0

// 0 zum abschalten. kennzeichnet die Laenge der Freeze Periode bei jedem Start.
mp_freezetime 2

// Zeit, in der man noch nach Rundenbegin kaufen kann (float)
mp_buytime 0.5

//Geld das am Anfang einer neuen Map zur Verfuegung steht
mp_startmoney 3500

// Spectators erlauben? 1=an 0=aus
allow_spectators 1

// 1=Friendlyfire ist an! 0=aus
mp_friendlyfire 1

// lowlag Eintrag fuer Internet-Server, fuer LAN weglassen
mp_lowlag "1"

// Granaten verletzen eigenes Team 1=an 0=aus
mp_friendly_grenade_damage 1

 // Bei 2 Hostagekills vom Server kicken  0=aus
mp_hostagepenalty 2

// maximum von Spielern die ein Team mehr Ueberzaehlig sein kann. 0=aus
mp_limitteams 1

// Logge Chat Messages?
// verwende die Server Admins zum ausgeben der Chatnachrichten in den Log Daten.
mp_logmessages 1

// Dauer einer Runde in Minuten
// Minimum der Runde ist 1 Min. Maximum ist 9 Min. 1.5 ist 90 Sekunden
mp_roundtime 4

// Maximale Anzahl von Runden bis der naechste Mapwechsel erfolgt.
mp_maxrounds 200

// Maximale Dauer einer Map in Minuten
// (Zeit bis die naechste Map Automatisch geladen wird.)
mp_timelimit 40

// Laedt die naechste Map wenn ein Team so viele Gewinne erzielt hat. 0=aus
mp_winlimit 0

// Teamkills werden mit Aussetzen bestraft 1=an 0=aus
mp_tkpunish "0"

//Prozentsatz der Spieler die eine Map waehlen muessen bevor dies geladen wird. 0.0 bis 1.0
mp_mapvoteratio 0.5

// Sets the amount of time in minutes before resetting the game,
// including frags, weapons, money, and scores -
// good for clan practice rounds. Set to '0' to disable.
sv_restartround 0

sv_enableoldqueries 1

//---------------------- General HL Cvars-----------------------

// Sets the amount of time in seconds in between client logo sprays.
decalfrequency 60

// Ob der Spieler schaden nimmt, falls er von goesseren Hoehen springt/faellt 0=aus 1=an
mp_falldamage 1

// Clients duerfen das Spiel nicht pausieren 0=aus 1=an
pausable 0
```

```
// automatisches Zielen deaktiviert  1=aktiviert
sv_aim 0

// Cheats deaktiviert 1=aktiviert
sv_cheats 0

// maximale Geschwindigkeit der Spieler
sv_maxspeed 320

// maximale Datenrate die an den Client gesendet wird
//sv_maxrate 30000 waere maximal fuer ETH
sv_maxrate 15000

// minimale Datenrate die an den Client gesendet wird
sv_minrate 4000

// Wie lange jemand in der Luft bleiben kann.
sv_airaccelerate 2

//---------------------------- Rcon & Password Stuff --------------

 // WICHTIG! rcon Passwort fuer Fernadministrierung
rcon_password "marinebiol"

// Evtl. vergebenes Passwort fuer den Server
sv_password ""

//----------------------------  Banned.cfg ---------------

// In the event that you have a llist of banned clients.
// this executes the banned.cfg file so that they will
// remain banned even after map changes.
exec banned.cfg

//------------------------- Cheating-Death -----------------------------

// Bei verwendung von Cheating-Death cdrequired 1,
//cdrequiredmsg,cdoptionalmsg,cdupdatemsg das // davor endfernen
// beziungsweise mit // auskommentiren bei nicht verwendung von Cheating-Death

// CD per rcon an,-ausmachen 0=optionat 1=an -1=aus
//cdrequired 1

// CD Req Nachricht:
//cdrequiredmsg "Cheating-Death ist Zwang!"

// CD optional Nachricht:
//cdoptionalmsg "Cheating-Death ist optional."

// CD Versions Update Nachricht:
//cdupdatemsg "Bitte dein Cheating-Death updaten!"

//---------------------------- server logging.----------------------

// Logfiles werden geschrieben ("+log on" in der Kommandozeile) 1=an 0=aus
mp_logfile 1
log on

//benutzt CVAR um den Level der anzeige einzustellen.
//( 0 = log enemy attacks. 1 = log Teammate attacks)
mp_logdetail 1

//soll ein logfile erstellt werden, oder eins per map?
sv_log_onefile 1


//---------------------------- Bot control----------------------

//sollen Bots sprechen?
bot_chatter off

//Bot skill level
```

```
bot_difficulty 2

//Duerfen Bots ohne Menschen spielen?
bot_join_after_player 0

//Wieviele Bots?
bot_quota 4

bot_quota_mode fill

bot_auto_vacate 1
```

A.6 server.cfg for ETH3

```
// -----------------------  Server Name (Hostname) -----------------

// hostname des Servers
hostname "ETH 3: Ballistic Geeks (No DynPricing)"

//-------------------------regionaleinstelung-------------------------
// Must specify sv_region, or it won't show up in Steam server browser

// 0: US East coast
// 1: US West coast
// 2: South America
// 3: Europe
// 4: Asia
// 5: Australia
// 6: Middle East
// 7: Africa
// 255:

sv_region 3


//-------------------------- CS Specific Cvars ----------------------

//Dynamic Pricing (1=on)
mp_dynamicpricing 0

// Informationen ueber andere Spieler in der Statusbar
// 0=alle 1=team 2=keine
mp_playerid 0

// Autokick fuer "idle Clients" & Ban Teamkiller? 1=an 0=aus
mp_autokick 0

// Einschalten der automatischen Balance der Teams. 1=an 0=aus
mp_autoteambalance "1"

// Dauer in Sekunden bis C4 explodiert
// Range is 15 - 90. =sekunden
mp_c4timer 35

// Taschenlampe darf benutzt werden 1=an 0=aus
mp_flashlight "1"

// Footsteps an? 1=an 0=aus
mp_footsteps 1

// erzwinge Verfolgerkamera?
// 0=freie Sicht 1=nur eigenes Team 3=Ort an dem man stirbt bleibt fix
mp_forcechasecam 1

// schwarzer Bildschirm nachdem man stirbt? 1=an 0=aus
mp_fadetoblack 0
```

```
// 0 zum abschalten. kennzeichnet die Laenge der Freeze Periode bei jedem Start.
mp_freezetime 2

// Zeit, in der man noch nach Rundenbegin kaufen kann (float)
mp_buytime 0.5

//Geld das am Anfang einer neuen Map zur Verfuegung steht
mp_startmoney 3500

// Spectators erlauben? 1=an 0=aus
allow_spectators 1

// 1=Friendlyfire ist an! 0=aus
mp_friendlyfire 1

// lowlag Eintrag fuer Internet-Server, fuer LAN weglassen
mp_lowlag "1"

// Granaten verletzen eigenes Team 1=an 0=aus
mp_friendly_grenade_damage 1

 // Bei 2 Hostagekills vom Server kicken  0=aus
mp_hostagepenalty 2

// maximum von Spielern die ein Team mehr ueberzaehlig sein kann. 0=aus
mp_limitteams 1

// Logge Chat Messages?
// verwende die Server Admins zum ausgeben der Chatnachrichten in den Log Daten.
mp_logmessages 1

// Dauer einer Runde in Minuten
// Minimum der Runde ist 1 Min. Maximum ist 9 Min. 1.5 ist 90 Sekunden
mp_roundtime 4

// Maximale Anzahl von Runden bis der naechste Mapwechsel erfolgt.
mp_maxrounds 200

// Maximale Dauer einer Map in Minuten
// (Zeit bis die naechste Map Automatisch geladen wird.)
mp_timelimit 40

// Laedt die naechste Map wenn ein Team so viele Gewinne erzielt hat. 0=aus
mp_winlimit 0

// Teamkills werden mit Aussetzen bestraft 1=an 0=aus
mp_tkpunish "0"

//Prozentsatz der Spieler die eine Map waehlen muessen bevor dies geladen wird. 0.0 bis 1.0
mp_mapvoteratio 0.5

// Sets the amount of time in minutes before resetting the game,
// including frags, weapons, money, and scores -
// good for clan practice rounds. Set to '0' to disable.
sv_restartround 0

sv_enableoldqueries 1

//---------------------- General HL Cvars-----------------------

// Sets the amount of time in seconds in between client logo sprays.
decalfrequency 60

// Ob der Spieler schaden nimmt, falls er von groesseren Hoehen springt/faellt 0=aus 1=an
mp_falldamage 1

// Clients duerfen das Spiel nicht pausieren 0=aus 1=an
pausable 0

// automatisches Zielen deaktiviert  1=aktiviert
sv_aim 0
```

```
// Cheats deaktiviert 1=aktiviert
sv_cheats 0

// maximale Geschwindigkeit der Spieler
sv_maxspeed 320

// maximale Datenrate die an den Client gesendet wird
sv_maxrate 15000

// minimale Datenrate die an den Client gesendet wird
sv_minrate 4000

// Wie lange jemand in der Luft bleiben kann.
sv_airaccelerate 2

//---------------------------- Rcon & Password Stuff --------------

 // WICHTIG! rcon Passwort fuer Fernadministrierung
rcon_password "marinebiol"

// Evtl. vergebenes Passwort fuer den Server
sv_password ""

//----------------------------  Banned.cfg ---------------

// In the event that you have a llist of banned clients.
// this executes the banned.cfg file so that they will
// remain banned even after map changes.
exec banned.cfg

//------------------------- Cheating-Death -----------------------------

// Bei verwendung von Cheating-Death cdrequired 1,
//cdrequiredmsg,cdoptionalmsg,cdupdatemsg das // davor endfernen
// beziungsweise mit // auskommentiren bei nicht verwendung von Cheating-Death

// CD per rcon an,-ausmachen 0=optionat 1=an -1=aus
//cdrequired 1

// CD Req Nachricht:
//cdrequiredmsg "Cheating-Death ist Zwang!"

// CD optional Nachricht:
//cdoptionalmsg "Cheating-Death ist optional."

// CD Versions Update Nachricht:
//cdupdatemsg "Bitte dein Cheating-Death updaten!"

//---------------------------- server logging.----------------------

// Logfiles werden geschrieben ("+log on" in der Kommandozeile) 1=an 0=aus
mp_logfile 1
log on

//benutzt CVAR um den Level der anzeige einzustellen.
//( 0 = log enemy attacks. 1 = log Teammate attacks)
mp_logdetail 1

//soll ein log file erstellt werden, doer eins per map?
sv_log_onefile 1


//---------------------------- Bot control----------------------

//sollen Bots sprechen?
//bot_chatter off

//Bot skill level
//bot_difficulty 2

//Duerfen Bots ohne Menschen spielen?
//bot_join_after_player 0
```

```
//Wieviele Bots?
//bot_quota 4

//bot_quota_mode fill

//bot_auto_vacate 1
```

Appendix B

Online Monitoring and Server Management Scripts

B.1 Server Start/Stop/Restart ”css.sh”

```
#!/bin/bash

#Author: Lorenz Breu
# small script to start, stop or restart the three css servers for my semester thesis
# Usage: css.sh {start|restart|stop} {1|2|3}

ARGS=2
ERR_BADARGS=65
ERR_NOSUCHSERVER=66
ERR_SCREENFAIL=67
ERR_KILLFAIL=68
BASE_DIR="/usr/local/games/hldsXX/"
SCRIPT="./srcds_run"
PARAMS="-game cstrike +maxplayers 14 +map cs_italy +ip 82.130.102.202 \
+port 27015 -tickrate 66 -autoupdate"

# ----------------------------------------------------------------------
# Test for correct number of args
if [ $# -ne 2 ]
then
echo "Usage: `basename $0` {start|restart|stop} {1|2|3}"
exit $ERR_BADARGS
fi

# ----------------------------------------------------------------------

# ----------------------------------------------------------------------
# Extract server number and prepare variables accordingly

case $2 in
"1")
PORT=27015
SCREEN="css1"
CURR_DIR=${BASE_DIR/XX/1}
CURR_PARAMS=${PARAMS/27015/$PORT}
;;

"2")
PORT=27016
MAP=de_dust2
SCREEN="css2"
CURR_DIR=${BASE_DIR/XX/2}
CURR_PARAMS=${PARAMS/27015/$PORT}
CURR_PARAMS=${CURR_PARAMS/cs_italy/$MAP}
;;

"3")
```

```
PORT=27017
SCREEN="css3"
CURR_DIR=${BASE_DIR/XX/3}
CURR_PARAMS=${PARAMS/27015/$PORT}
;;

*)
echo "No CSS server with that number available, try 1-3..."
exit $ERR_NOSUCHSERVER
;;
esac
# ---------------------------------------------------------------------

# ---------------------------------------------------------------------
# process option

case "$1" in
"start")
if [[ `screen -ls |grep $SCREEN` ]]
then
        echo "CSS Server $2 is already running, \
         type \"`basename $0` restart $2\" to restart Server $2"
exit 0
else
echo "Starting Counter-Strike Server $2..."
cd $CURR_DIR
screen -d -m -S $SCREEN $SCRIPT $CURR_PARAMS
if [ "$?" -eq 0 ]
then
echo "Server started..."
exit 0
else
echo "Failed to start server..."
exit $ERR_SCREENFAIL
fi
fi
;;

"stop")
if [[ `screen -ls | grep $SCREEN` ]]
then
echo "Stopping server $2..."
kill `screen -ls |grep $SCREEN |awk -F . '{print $1}'|awk '{print $1}'`
if [ "$?" -eq 0 ]
then
echo "Server stopped..."
exit 0
else
echo "Could not kill Server..."
exit $ERR_KILLFAIL
fi
else
echo "Server $2 is not currently running..."
exit 0
fi
;;

"restart")
if [[ `screen -ls | grep $SCREEN` ]]
then
echo "Restarting Server $2..."
echo "Phase 1: stopping running server..."
kill `screen -ls |grep $SCREEN |awk -F . '{print $1}'|awk '{print $1}'`
if [ "$?" -ne 0 ]
then
echo "Could not kill Server..."
exit $ERR_KILLFAIL
fi
echo "Server successfully stopped..."
echo "Phase 2: Starting server $2..."
cd $CURR_DIR
screen -d -m -S $SCREEN $SCRIPT $CURR_PARAMS
if [ "$?" -ne 0 ]
```

```
then
echo "Failed to start server..."
exit $ERR_SCREENFAIL
fi
echo "Server successfully started..."
exit 0
else
echo "Server $2 not currently running..."
exit 0
fi
;;

*)
echo "Unknown command"
echo "Usage: `basename $0` {start|restart|stop} {1|2|3}"
exit $ERR_BADARGS
;;
esac


exit 0
```

B.2 Player/CPU Load Monitoring ”online_monitoring.py”

The following Python 2.5 script was used to monitor player and cpu load, store the data in RRD’s and produce images for the webpage.

```
import rrdtool
import SRCDS as rcon
from optparse import OptionParser
import os
import string
import time
import ping


class MyServer:
def __init__(self,server_port=27015,pwd='marinebiol'):
self.port=server_port
self.rcon_pwd=pwd

basedir=''
rrdfile=''
logfile='/home/srcds/logs/pings/icmp_ping.dmp'
portlist=''
htdocs='/var/www/'
statfile='/proc/stat'
host='82.130.102.202'
create_file=False
verbose=False
old_total=0.0
old_idle=0.0
f=''

def countPlayers(list):
result = []
for i in list:
if i['time_on']>0:
result.append(i)
return len(result)

def listPlayers(list):
result=[]
for i in list:
if i['time_on']>0:
result.append(i)
```

```
return result

def getLoad():
global old_total
global old_idle
result=0.0
fields=[]
file=open(statfile)
line=file.readline()
fields=string.split(line)
idle=float(fields[4])
delta_idle=idle-old_idle
system=float(fields[1])
user=float(fields[2])
nice=float(fields[3])
total=user+system+nice+idle
delta_total=total-old_total
result=(1.0-(delta_idle/delta_total))*100
file.close()
old_total=total
old_idle=idle
return result

def createRRD(file):
global basedir
rrdtool.create(file,'--step',
 '10','DS:srv27015:GAUGE:20:0:14','DS:srv27016:GAUGE:20:0:14','DS:srv27017:GAUGE:20:0:14',
 'DS:total:GAUGE:20:0:42','RRA:AVERAGE:0.5:1:259200')
rrdtool.create(basedir+'cpuload.rrd','--step', '10','DS:cpu:GAUGE:20:0:100',
 'RRA:AVERAGE:0.5:1:259200')

def parseInput():
global basedir
global rrdfile
global portlist
global gfxfiles
global create_file
global verbose
parser=OptionParser()
parser.set_defaults(basedir='/home/srcds/logs/rrd/')
parser.set_defaults(ports='27015,27016,27017')
parser.set_defaults(file='srcds.rrd')
parser.set_defaults(create=False)
parser.set_defaults(verb=False)
parser.add_option('-v',action='store_true',dest='verb')
parser.add_option('-d',action='store',type='string',dest='basedir')
parser.add_option('-p',action='store',type='string',dest='ports')
parser.add_option('-f',action='store',type='string',dest='file')
parser.add_option('-c',action='store_true',dest='create')
(options, args) = parser.parse_args()
basedir=options.basedir
rrdfile=basedir+options.file
portlist=options.ports.split(',')
create_file=options.create
verbose=options.verb


def makeServerList():
result=[]
for p in portlist:
s=MyServer(server_port=int(p))
result.append(s)
return result

def makeUpdates(servers):
global host
f=open(logfile,'a')
updateString='N'
total=0
for s in servers:
if verbose:
print 'connecting to ' + host + ' on port %i with password %s' % (s.port, s.rcon_pwd)
rconServer=rcon.SRCDS(host,port=s.port,rconpass=s.rcon_pwd)
```

```
i=countPlayers(rconServer.players())
if i>14:
i=0
tempString=':%i' % (i)
updateString=updateString+tempString
total=total+i
stat=rconServer.status()
for j in stat:
if verbose:
print 'pinging '+j['adr'][:-6]
print 'server ping: '+str(j['ping'])
sp=j['ping']
r = ping.doOne(j['adr'][:-6],1)
if r!=None:
ap=int(1000*r)
if verbose:
print 'active ping: '+str(ap)
print 'Updating logfile with measured pings...'

else:
ap=-1
if verbose:
print 'active ping: None'
print 'Updating logfile with measured pings...'
t=repr(time.time())
f.write(t+' '+j['adr']+' '+str(j['uniqueid'])+' '+str(sp)+' '+str(ap)+' '+str(j['loss']))
tempString=':%i' % (total)
updateString=updateString+tempString
if verbose:
print "Updating "+rrdfile+"..."
print updateString
rrdtool.update(rrdfile,updateString)
load=getLoad()
updateString='N:%f' % (load)
if verbose:
print "Updating "+basedir+"cpuload.rrd..."
print updateString
rrdtool.update(basedir+'cpuload.rrd',updateString)
if verbose:
print "done"
f.close()



parseInput()
f=open(logfile,'r')
if f.readline()=='':
f=open(logfile,'a')
f.write('#time ip steamid srcds_ping icmp_ping pkt_loss')
f.close()
if create_file:
print 'Creating RRD file in '+basedir
createRRD(rrdfile)
else:
serverlist=makeServerList()
gfxfiles_3600=[htdocs+'images/out_eth1.png',htdocs+'images/out_eth2.png',htdocs+
'images/out_eth3.png']
gfxfiles_86400=[htdocs+'images/out_eth1_24h.png',htdocs+'images/out_eth2_24h.png',htdocs+
'images/out_eth3_24h.png']
gfxfiles_604800=[htdocs+'images/out_eth1_1w.png',htdocs+'images/out_eth2_1w.png',htdocs+
'images/out_eth3_1w.png']
while True:
makeUpdates(serverlist)
if verbose:
print 'graphing from '+rrdfile
rrdtool.graph(gfxfiles_3600[0],'--end','now','--start','end-3600s','--title','Players on
 ETH1','--width','350','DEF:srv27015='+rrdfile+':srv27015:AVERAGE','LINE2:srv27015#FF0000:\'ETH1\'')
rrdtool.graph(gfxfiles_3600[1],'--end','now','--start','end-3600s','--title','Players on
 ETH2','--width','350','DEF:srv27016='+rrdfile+':srv27016:AVERAGE','LINE2:srv27016#FF0000:\'ETH2\'')
rrdtool.graph(gfxfiles_3600[2],'--end','now','--start','end-3600s','--title','Players on
 ETH3','--width','350','DEF:srv27017='+rrdfile+':srv27017:AVERAGE','CDEF:filter=srv27017,14,
 GT,0,srv27017,IF','LINE2:filter#FF0000:\'ETH3\'')
```

```
rrdtool.graph(htdocs+'images/out_cpu.png','--end','now','--start','end-3600s','--title',
 'Total CPU Load in %','--width','350','DEF:cpu='+basedir+'cpuload.rrd:cpu:AVERAGE',
 'LINE2:cpu#FF0000:\'CPU Load\'')

rrdtool.graph(gfxfiles_86400[0],'--end','now','--start','end-86400s','--title','Players on
 ETH1','--width','350','DEF:srv27015='+rrdfile+':srv27015:AVERAGE','LINE2:srv27015#FF0000:\'ETH1\'')
rrdtool.graph(gfxfiles_86400[1],'--end','now','--start','end-86400s','--title','Players on
 ETH2','--width','350','DEF:srv27016='+rrdfile+':srv27016:AVERAGE','LINE2:srv27016#FF0000:\'ETH2\'')
rrdtool.graph(gfxfiles_86400[2],'--end','now','--start','end-86400s','--title','Players on
 ETH3','--width','350','DEF:srv27017='+rrdfile+':srv27017:AVERAGE','CDEF:filter=srv27017,14,
 GT,0,srv27017,IF','LINE2:filter#FF0000:\'ETH3\'')
rrdtool.graph(htdocs+'images/out_cpu_24h.png','--end','now','--start','end-86400s','--title',
 'Total CPU Load in %','--width','350','DEF:cpu='+basedir+'cpuload.rrd:cpu:AVERAGE',
 'LINE2:cpu#FF0000:\'CPU Load\'')

rrdtool.graph(gfxfiles_604800[0],'--end','now','--start','end-604800s','--title','Players on
ETH1','--width','350','DEF:srv27015='+rrdfile+':srv27015:AVERAGE','LINE2:srv27015#FF0000:\'ETH1\'')
rrdtool.graph(gfxfiles_604800[1],'--end','now','--start','end-604800s','--title','Players on
ETH2','--width','350','DEF:srv27016='+rrdfile+':srv27016:AVERAGE','LINE2:srv27016#FF0000:\'ETH2\'')
rrdtool.graph(gfxfiles_604800[2],'--end','now','--start','end-604800s','--title','Players on
ETH3','--width','350','DEF:srv27017='+rrdfile+':srv27017:AVERAGE','CDEF:filter=srv27017,14,
GT,0,srv27017,IF','LINE2:filter#FF0000:\'ETH3\'')
rrdtool.graph(htdocs+'images/out_cpu_1w.png','--end','now','--start','end-604800s','--title',
'Total CPU Load in %','--width','350','DEF:cpu='+basedir+'cpuload.rrd:cpu:AVERAGE',
'LINE2:cpu#FF0000:\'CPU Load\'')

info=rrdtool.info(rrdfile)
if verbose:
print info['last_update']
time.sleep(10-(time.time()%10))
```

B.3 Keep Monitoring Alive ”logcontrol.sh”

The following script was run as a cronjob every 5 minutes to ensure monitoring was up and running.

```
#!/bin/bash

SCRIPT_DIR="/home/srcds/upload/python/"
LOGERROR=65

if [[ `screen -ls | grep log` ]]
then
echo "server_cpu.py seems to be running..."
exit 0
else
echo "server_cpu seems to be down, restarting..."
cd $SCRIPT_DIR
screen -d -m -S log python online_monitoring.py -v
if [ "$?" -eq 0 ]
then
echo "logging started..."
exit 0
else
echo "Failed to start logging..."
echo "error $?"
exit $LOGERROR
fi
exit 0
fi
exit 0
```

Appendix C

Original Problem

The following is the original problem as published by the ETH, in particular the TIK group
Due to the recent global explosion of on-line multi-player gaming, it is becoming more important to understand its network behavior and usage in order to provision and design future network infrastructure. With the launches of Microsoft?s Xbox2 and future Playstation on-line game networks and with the emergence of massively multi-player on-line games, it is clear that a large increase in gaming traffic is imminent. In order to understand the characteristics of Internet gaming, we will examine the behavior of an popular game server, specifically a Counter-Strike server.
The goal of this thesis is to setup a game server together with a measurement system and to trace and analyze the generated network traffic. From the packet traces, we also plan to determine patterns or signatures that can be used to filter on-line game traffic from the NetFlow data collected in the DDoSVax project. More information about the DDoSVax project and the available infrastructure can be found on the project homepage.
Since most of our infrastructure runs Linux, some development experience under Linux would be beneficial. We use C for most of our implementations for speed reasons and Perl for prototyping, but we are flexibel as to other approaches.

Bibliography

[1] Fatal1ty.com, *"Fatal1ty.com"*, http://www.fatal1ty.com/

[2] SwissQuake, *"SwissQuake.ch - HiSpeed & LowPing Gameservers"*, http://www.swissquake.ch/

[3] GameCenter, *"GAME CENTER - swiss gameservers - Home"*, http://www.gamecenter.ch/

[4] McCreary, S. and K. Claffy, "Trends in wide area IP traffic patterns - A view from Ames Internet Exchange", in *ITC Specialist Seminar*. 2000.

[5] Dick, M., O. Wellnitz, and L. Wolf. "Analysis of Factors Affecting Players' Performance and perception in Multiplayer Games". in *Proceedings of the 4th ACM SIGCOMM workshop on Network and system support for games Netgames '05*. 2005.

[6] Borella, M.S., "Source Models of Network Game Traffic", in *Networld and Interop '99 Engineer's Conference*. 1999.

[7] Chang, F. and W.-c. Feng. "Modeling Player Session Times of On-line Games". in *Proceedings of the 2nd workshop on Network and system support for games NetGames '03*. 2003.

[8] Färber, J. "Network Game Traffic Modelling". in *Proceedings of the 1st workshop on Network and system support for games NetGames '02*. 2002.

[9] Claypool, M., D. LaPoint, and J. Winslow. "Network analysis of Counter-strike and Starcraft". in *Performance, Computing, and Communications Conference*. 2003.

[10] Feng, W.-c., et al., "A Traffic Characterization of Popular On-line Games". *IEEE/ACM Transactions on Networking (TON)*, vol. 13, no. 3, 2005.

[11] Steam, *"Welcome to Steam"*, http://www.steampowered.com.

[12] Steam, *"Steam: Game and Player Statistics"*, http://www.steampowered.com/v/index.php?area=stats (Accessed 03/07/07)

[13] Valve, *"Valve"*, http://www.valvesoftware.com/.

[14] EA Link Homepage, *"EA Link - Computer Games, Video Games, and Online PC Game Downloads"*, http://www.ea.com/ealink/index.jsp.

[15] GameSpy Arcade, *"GameSpy Arcade - Play Hundreds of Online Multiplayer Games!"*, http://www.gamespyarcade.com/.

[16] Färber, J., "Traffic Modelling for Fast Action Network Games". *Multimedia Tools and Applications*, vol. 23, no. 1, pp. 31-46, 2004.

[17] Branch, P. and G. Armitage. "Extrapolating Server to Client IP traffic From Empirical Measurements of First Person Shooter games". in *Proceedings of the 5th ACM SIGCOMM workshop on Network and system support for games*. 2006.

[18] McLeod, R. "A Traffic Analysis of a Small Private Network Compromised by an On-line Gaming Host". in *Proceedings of FloCon 2006*. 2006.

[19] But, J., et al. "Automated Network Games Enhancement Layer - A Proposed Architecture". in *Proceedings of 5th ACM SIGCOMM workshop on Network and system support for games*. 2006.

[20] Dübendorfer, T., A. Wagner, and B. Plattner. "A Framework for Real-Time Worm Attack Detection and Backbone Monitoring". in *Proceedings of the First IEEE International Workshop on Critical Infrastructure Protection IWCIP '05*. 2005.

[21] Wagner, A., et al. "Experiences with Worm Propagation Simulations". in *Proceedings of the 2003 ACM workshop on Rapid malcode WORM '03*. 2003.

[22] DDoSVax Project Homepage, *"DDoSVax"*, http://www.tik.ee.ethz.ch/ ddosvax/.

[23] Mani's Admin Plugin Homepage, *"Mani's Admin Plugin - Home"*, http://www.mani-admin-plugin.com/.

[24] HLSW homepage, *"HLSW :: Game server browser and administration tool"*, http://www.hlsw.net/.

[25] Valve Developer Community Wiki, *"Source Multiplayer Networking - Valve Developer Community"*, http://developer.valvesoftware.com/wiki/Source_Multiplayer_Networking/.

[26] ntop Project Homepage, *"Welcome to ntop.org"*, http://www.ntop.org.

[27] World of Warcraft Europe Homepage, *"World of Warcraft Europe"*, http://www.wow-europe.com/de/index.xml.

[28] Maxmind Homepage, *"MaxMind - IP Geolocation and Online Fraud Prevention"*, http://www.maxmind.com/.

[29] Entertainment Software Association Homepage, *"ESA"*, http://www.theesa.com/.

[30] PricewaterhouseCoopers, *"PricewaterhouseCoopers Says Entertainment and Media Industry in Solid Growth Phase, Will Grow 6.6 Percent Annually to $1.8 Trillion in 2010"*, http://www.pwc.com/extweb/ncpressrelease.nsf/docid/283F75E5D932C00385257194004DDD0A, 2006

[31] DFC Intelligence, *"Who Will Benefit from the Growth of Online Game Subscription Revenue?"*, http://www.dfcint.com/game_article/mar06article.html.

[32] Ethereal Homepage, *"Ethereal: A Network Protocol Analyzer"*, http://www.ethereal.com/.

[33] Teamspeak Homepage, *"TeamSpeak"*, http://www.teamspeak.com/

[34] Schatzmann, D. Analyzing network traffic from the SWITCH network from 2003 until today". in *ETH Master Thesis MA-2007-12*. 2007.

[35] Cstrike-Planet Installation Tutorial, *"Counter-Strike » Linux: Install CS Source"*, http://www.cstrike-planet.com/tutorial/1-Linux-Install-CS-Source/5/.