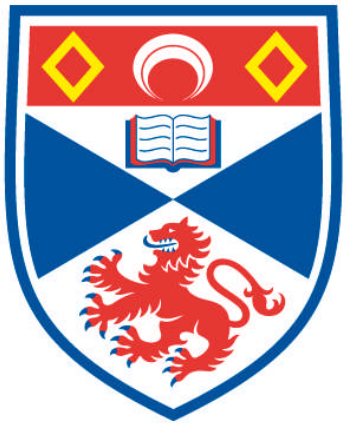
ADAPTIVE NETWORK TRAFFIC MANAGEMENT FOR MULTI USER VIRTUAL ENVIRONMENTS

Iain Oliver

A Thesis Submitted for the Degree of PhD
at the
University of St Andrews



2011

Full metadata for this item is available in
Research@StAndrews:FullText
at:
http://research-repository.st-andrews.ac.uk/

Please use this identifier to cite or link to this item:
http://hdl.handle.net/10023/3090

This item is protected by original copyright

Adaptive Network Traffic Management for Multi User Virtual Environments

Iain Oliver

July 1, 2011

Abstract

Multi User Virtual Environments (MUVE) are a new class of Internet application with a significant user base. This thesis adds to our understanding of how MUVE network traffic fits into the mix of Internet traffic, and how this relates to the application's needs.

MUVEs differ from established Internet traffic types in their requirements from the network. They differ from traditional data traffic in that they have soft real-time constraints, from game traffic in that their bandwidth requirements are higher, and from audio and video streaming traffic in that their data streams can be decomposed into elements that require different qualities of service. This work shows how real-time adaptive measurement based congestion control can be applied to MUVE streams so that they can be made more responsive to changes in network conditions than other real-time traffic and existing MUVE clients. It is shown that a combination of adaptive congestion control and differential Quality of Service (QoS) can increase the range of conditions under which MUVEs both get sufficient bandwidth and are Transport Control Protocol (TCP) fair.

The design, implementation and evaluation of an adaptive traffic management system is described. The system has been implemented in a modified client, which allows the MUVE to be made TCP fair without changing the server.

1. Candidate's declarations:

I, Iain Oliver, hereby certify that this thesis, which is approximately 84,740 words in length, has been written by me, that it is the record of work carried out by me and that it has not been submitted in any previous application for a higher degree.

I was admitted as a research student in [month, year] and as a candidate for the degree of Ph.D. in September, 2006; the higher study for which this is a record was carried out in the University of St Andrews between 2006 and 2010.

Date 09/08/2011 signature of candidate ……

2. Supervisor's declaration:

I hereby certify that the candidate has fulfilled the conditions of the Resolution and Regulations appropriate for the degree of Ph.D. in the University of St Andrews and that the candidate is qualified to submit this thesis in application for that degree.

Date 09/08/2011 signature of supervisor ……

3. Permission for electronic publication: (*to be signed by both candidate and supervisor*)

In submitting this thesis to the University of St Andrews I understand that I am giving permission for it to be made available for use in accordance with the regulations of the University Library for the time being in force, subject to any copyright vested in the work not being affected thereby. I also understand that the title and the abstract will be published, and that a copy of the work may be made and supplied to any bona fide library or research worker, that my thesis will be electronically accessible for personal or research use unless exempt by award of an embargo as requested below, and that the library has the right to migrate my thesis into new electronic forms as required to ensure continued access to the thesis. I have obtained any third-party copyright permissions that may be required in order to allow such access and migration, or have requested the appropriate embargo below.

The following is an agreed request by candidate and supervisor regarding the electronic publication of this thesis:

*Add **one** of the following options:*

(i) Access to printed copy and electronic publication of thesis through the University of St Andrews.

Date 09/08/2011 signature of candidate …… signature of supervisor ……

A supporting statement for a request for an embargo must be included with the submission of the draft copy of the thesis. Where part of a thesis is to be embargoed, please specify the part and the reasons.

Contents

1 Introduction **1**
- 1.1 Overview . . . 1
- 1.2 Goals of the work . . . 4
- 1.3 Thesis Statement . . . 4
- 1.4 Road Map of thesis . . . 5
- 1.5 Summary . . . 6

2 Internet traffic and Congestion Control **7**
- 2.1 Internet . . . 7
  - 2.1.1 Transmission Control Protocol . . . 8
  - 2.1.2 Congestion . . . 9
- 2.2 Congestion Control . . . 9
  - 2.2.1 Window and rate based congestion control . . . 10
  - 2.2.2 TCP congestion control . . . 10
  - 2.2.3 Modelling TCP Congestion Control . . . 12
  - 2.2.4 Other Congestion Controlled Transport Protocols . . . 14
  - 2.2.5 Summary . . . 15
- 2.3 Internet Applications and Traffic Types . . . 15
- 2.4 Summary . . . 17

3 Methodology **19**
- 3.1 Packet Capture . . . 19
- 3.2 Analysis program: SLparse . . . 20
  - 3.2.1 Related work . . . 21
  - 3.2.2 DCCP . . . 22
  - 3.2.3 Second Life . . . 22
  - 3.2.4 Passive loss estimation . . . 22
  - 3.2.5 Passive RTT estimation . . . 22
  - 3.2.6 Summary . . . 23
- 3.3 NetEM . . . 23
- 3.4 Process of this work . . . 24
- 3.5 Summary . . . 26

4 Multi User Virtual Environments **27**
- 4.1 MUVE History . . . 27
- 4.2 MUVE Definition . . . 28
  - 4.2.1 Example MUVEs . . . 28
  - 4.2.2 Taxonomy of MUVEs . . . 30
  - 4.2.3 Definition . . . 31
  - 4.2.4 Summary . . . 31
- 4.3 Second Life . . . 32
  - 4.3.1 Second Life Structure . . . 33
  - 4.3.2 Circuits . . . 34

4.3.3 Packets . . . 34
4.3.4 Adaptation to network conditions . . . 35
4.3.5 Channels . . . 35
4.3.6 Avatar Control and Reaction . . . 37
4.3.7 Summary . . . 38
4.4 Comparison with Games Traffic . . . 38
4.4.1 MMORPG . . . 38
4.4.2 Third Person Omnipresent . . . 39
4.4.3 First Person Shooters . . . 39
4.4.4 Summary . . . 40
4.5 MUVE . . . 40
4.6 Conclusion . . . 42

5 Second Life Evaluation **44**
5.1 Related Work . . . 44
5.1.1 Measurement of SL Traffic . . . 44
5.1.2 Comparison with Games . . . 45
5.2 Observing SL in the wild . . . 46
5.2.1 Methodology . . . 47
5.2.2 Experimental aims . . . 47
5.2.3 Network Conditions . . . 48
5.2.4 Summary Statistics . . . 49
5.2.5 Packets . . . 51
5.2.6 Circuits . . . 52
5.2.7 Channels . . . 53
5.2.8 Summary . . . 55
5.2.9 Conclusion . . . 55
5.3 Experimental Investigation . . . 57
5.3.1 Methodology . . . 58
5.3.2 SWFT: Standing Walking, Flying, Teleporting . . . 59
5.3.3 Representation of Avatar's Position . . . 61
5.3.4 Decomposition by Channel . . . 63
5.3.5 The Task Channel and Avatar Control . . . 64
5.3.6 The Second Life Client Cache . . . 65
5.3.7 Throttle Settings . . . 66
5.3.8 Throttle Adaptation . . . 68
5.3.9 Summary . . . 69
5.3.10 Conclusion . . . 70
5.4 Second Life traffic requirements . . . 70
5.5 Summary . . . 72

6 Adaptive traffic management for MUVEs **73**
6.1 Comparison of SL and TCP adaptive traffic management . . . 74
6.1.1 TCP . . . 74
6.1.2 Validating the TCP fair equation . . . 74
6.1.3 Second Life traffic management . . . 75
6.1.4 Comparison of SL and TCP fairness . . . 76
6.1.5 Summary of the comparison of Second Life and TCP adaptation systems . . . 78
6.2 Design of Control Algorithm . . . 78
6.2.1 Window tracking systems . . . 78
6.2.2 Formula controlled systems . . . 79
6.2.3 Design Choices for traffic management system . . . 80
6.3 Mongoose structure . . . 82
6.3.1 The throttle calculation system . . . 84

6.3.2 Summary of Mongoose structure . . . 89
6.4 Summary . . . 89

7 Evaluation **90**
7.1 Methodology . . . 90
7.2 Calculated throttle values . . . 92
7.3 Comparison of the Mongoose window tracking system with TFRC. . . . 97
7.4 Bandwidth utilization against requested bandwidth. . . . 100
7.4.1 Calculation interval . . . 102
7.5 Bandwidth utilization and TCP fair rate. . . . 105
7.6 Conclusion . . . 107

8 Traffic Management with OpenSim **108**
8.1 Traffic management system . . . 108
8.2 OpenSim traffic examination . . . 109
8.2.1 Methodology . . . 110
8.2.2 Summary statistics and analysis by protocol of an OpenSim session. . . . 111
8.2.3 Standing, Walking, Flying and Teleporting . . . 112
8.2.4 Islands . . . 114
8.2.5 Packet sizes . . . 115
8.2.6 Channels . . . 116
8.2.7 Summary . . . 117
8.3 Throttle system and network conditions. . . . 117
8.3.1 Second Life client and throttle settings. . . . 117
8.3.2 Second Life client and loss levels. . . . 119
8.3.3 Mongoose client and loss levels. . . . 120
8.3.4 Per channel throttle values. . . . 123
8.3.5 Summary . . . 123
8.4 OpenSim . . . 123
8.4.1 OpenSim Packet throttle system. . . . 124
8.4.2 Modified OpenSim server . . . 125
8.4.3 Summary . . . 126
8.5 Measurement of Modified OpenSim server. . . . 126
8.5.1 Per channel throttle values. . . . 127
8.5.2 Total throttle burst size. . . . 127
8.5.3 Mongoose client and loss levels with the modified client. . . . 129
8.5.4 Throttle and bandwidth against loss rate. . . . 131
8.5.5 Summary . . . 131
8.6 Comparison between different versions. . . . 132
8.7 Conclusion . . . 133

9 Summary and Conclusions **134**
9.1 Traffic management properties . . . 134
9.2 Summary of Work . . . 136
9.2.1 Goals . . . 136
9.2.2 Summary of Chapters . . . 136
9.2.3 Summary . . . 139
9.3 Contributions of the Work . . . 139
9.4 Limitations of the Work and Future work . . . 141
9.5 Thesis statement . . . 142
9.6 Conclusion . . . 142

A Paper **154**

B Server Side traffic management **155**
B.1 OpenSim over DCCP server and client . . . 155
B.2 Client and server communicating over DCCP . . . 156
B.3 Evaluation of TCP fair algorithms using DCCP . . . 158
B.4 Future Work . . . 160
B.5 Conclusion . . . 160

C User Level DCCP **161**
C.1 Datagram Congestion Control Protocol (DCCP) . . . 162
C.2 Userspace network protocol implementations . . . 163
C.3 Library . . . 164
C.3.1 Provided functionality . . . 164
C.3.2 Structure . . . 165
C.3.3 Congestion control system . . . 167
C.4 Process . . . 168
C.4.1 Socket creation . . . 168
C.4.2 Sending data . . . 169
C.4.3 Recive data . . . 170
C.4.4 Connection close . . . 170
C.4.5 Features . . . 170
C.4.6 Main loop . . . 171
C.4.7 DispatchCommand . . . 171
C.4.8 Decisions . . . 174
C.4.9 updateconnection . . . 174
C.4.10 Connection . . . 175
C.5 Evaluation . . . 175
C.6 Conclusion . . . 178

D Tables **180**

List of Figures

1.1 Second Life and OpenSim screen shots. . . . 3

2.1 A router where congestion is happening. . . . 9
2.2 Four different views of the TCP fair equation over a range of RTTs and loss regimes. . . . 13

3.1 The structure of the test network. . . . 20

4.1 Object creation and manipulation in Second Life. . . . 32
4.2 The structure of Second Life. . . . 33
4.3 The Second Life packet header structures 34
4.4 Second Life throttle settings. . . . 36
4.5 The amount bandwidth for each channel's throttle against the total. . . . 37

5.1 The structure of the lab network. . . . 46
5.2 Round trip time values. . . . 48
5.3 The aggregate packet and bytes per second. . . . 49
5.4 The amount bandwidth used by Machine 3. . . . 50
5.5 The number of reliable and unreliable packets per second against time. . . . 51
5.6 The distribution of packet sizes. . . . 52
5.7 The distribution of packet sizes for machine 1. . . . 53
5.8 The bandwidth used by each of the channel throttles. . . . 54
5.9 The bandwidth used by each of the throttles. . . . 56
5.10 Object creation in Second Life. . . . 57
5.11 The path taken by the avatar 59
5.12 Throughput Against Time 60
5.13 The ECDF of throughput for the two islands. . . . 60
5.14 The ECDF of Packet Size for the two islands. . . . 61
5.15 Positional Error 62
5.16 Decomposition by Channel 63
5.17 Avatar Traffic 64
5.18 Throttles 65
5.19 Effect of Caching 66
5.20 The bandwidth used by each of the throttles under different network conditions. . . . 67
5.21 The bandwidth used by each of the throttles under different network conditions. . . . 68
5.22 The bandwidth allocated to each of the throttles under different network conditions. . . . 69
5.23 SL Adaptation to Loss 70

6.1 3D graphs of TCP fair rate equation and measurements. . . . 75
6.2 2D graphs of TCP equation and measurements. . . . 76
6.3 SL Adaptation to Loss 77
6.4 Different loss rates. . . . 79
6.5 Calculating the sending rate after a change in loss rate. . . . 80
6.6 The existing system. . . . 81
6.7 The data-flow in the client and server. . . . 82
6.8 A call-graph of the client. . . . 83

6.9 The structure of an AgentThrottle packet. . . . 87

7.1 The structure of the test network. . . . 91
7.2 Loss levels against time. . . . 92
7.3 Second Life calculated throttle values and TFRC values at different loss levels. . . . 93
7.4 Mongoose calculated throttle values and TFRC values at different loss levels. . . . 94
7.5 Calculated throttle values against loss rate. . . . 95
7.6 Mongoose calculated throttle values and TFRC values with a change in loss rate. . . . 96
7.7 Simulations of the behaviour types of Congestion control system with different loss rates. . . . 97
7.8 Convergence time for Window tracking and formula based congestion control in simulations. . . . 98
7.9 Mongoose calculated throttle values and bandwidth utilization with different loss levels. . . . 100
7.10 Mongoose calculated throttle values, bandwidth utilization and TFRC values with different loss levels and a 5 second calculation interval. . . . 101
7.11 Requested bandwidth against bandwidth and the difference between bandwidth and the throttle. . . . 102
7.12 Difference between the requested bandwidth and used bandwidth against the calculation interval. . . . 104
7.13 Bandwidth usage against TCP fair value. . . . 105
7.14 Bandwidth against loss rate. . . . 106

8.1 The structure of the test network. . . . 110
8.2 Screen shots of the three islands used. . . . 110
8.3 The total amount of packet and bytes sent from an avatar walking around the cathedral island. . . . 111
8.4 The bandwidth usage and throttle values for different activities on the Sparta island. . . . 113
8.5 The throttle and bandwidth usage for walking on different islands. . . . 114
8.6 The distribution of packet sizes for the cathedral trace. . . . 116
8.7 Fraction of bandwidth per throttle for the Sparta island. . . . 117
8.8 Bandwidth usage against Throttle values. . . . 118
8.9 The bandwidth usage and throttle values for loss levels. . . . 119
8.10 Calculated throttle values and bandwidth usage against loss rate. . . . 120
8.11 The bandwidth usage and throttle values for loss levels. . . . 121
8.12 Calculated throttle values and bandwidth usage against loss rate. . . . 122
8.13 The throttle limiting bandwidth usage. . . . 122
8.14 The token bucket and queue throttle system. . . . 124
8.15 The relationship between the token buckets and queues in OpenSim TR. . . . 125
8.16 The throttle limiting bandwidth usage. . . . 127
8.17 Bandwidth usage against burst size as a fraction of the throttle value. . . . 128
8.18 The bandwidth usage and throttle values for loss levels. . . . 130
8.19 Calculated throttle values and bandwidth usage against loss rate. . . . 131
8.20 CDF of bandwidth for different combinations of the clients and servers. . . . 132

B.1 The overall structure of the system. . . . 156
B.2 The bandwidth utilisation against number of connections for the two DCCP implementations and both congestion control systems and TCP . . . 157
B.3 The number of bytes per second for one DCCP connections and one TCP connection, sampled once every 10 milliseconds, on a network with no routers. . . . 158
B.4 The number of bytes per second for one DCCP connections and one TCP connection, sampled once every 10 milliseconds, on a network with 1 router. . . . 159
B.5 The fairness between a TCP connection and a DCCP connection against the RTT of the link. . . . 159

C.1 The overall structure of DCCP . . . 165
C.2 The structure of the DCCP connection state struct . . . 166
C.3 The state transition diagram for DCCP . . . 168
C.4 The call graph of the bindconnection function . . . 168
C.5 The call graph of the activeopen function . . . 169
C.6 The call graph of the attemptsend function . . . 169
C.7 The feature state transition diagram for DCCP . . . 170

C.8 The state transition diagram for DCCP 171
C.9 The number of bytes per second for a library DCCP connection, sampled once a second. 174
C.10 The bandwidth utilisation against number of connections for the two DCCP implementations and both congestion control systems and TCP 176
C.11 The number of bytes per second for one DCCP connections and one TCP connection, sampled once every 10 milliseconds, on a network with no routers. 177
C.12 The number of bytes per second for one DCCP connections and one TCP connection, sampled once every 10 milliseconds, on a network with 1 router. 178
C.13 The fairness between a TCP connection and a DCCP connection against the RTT of the link. 178
C.14 The bandwidth utilisation of two DCCP connections travelling through the same router. 179
C.15 The bandwidth utilisation of a kernel level DCCP connection carrying an RTP stream of audio. 179

List of Tables

2.1 The 4 layer model. . . . 8
2.2 Type of network traffic. . . . 16

4.1 Comparison of MUVEs . . . 32

5.1 The average bandwidth usage of different type of game. . . . 45
5.2 The timetable for the lab session . . . 46
5.3 Summary statistics for all the machines. . . . 48
5.4 Summary statistics for individual machines. . . . 50
5.5 Summary statistics for the channels. . . . 53
5.6 Summary statistics for the channels average over the number of machines. . . . 54
5.7 Summary statistics for the channels for machine 1. . . . 55
5.8 Throughput for SWFT on different Islands. . . . 61
5.9 Error in SL meters at 150 ms RTT . . . 62

7.1 Throttle calculation statistics . . . 94

8.1 Summary statistics for a trace with the avatar walking around the cathedral island. . . . 111
8.2 Summary of bandwidth levels for the different traces. . . . 114

D.1 Information about packets. . . . 181
D.2 The number of connections associated with each machine. . . . 182
D.3 Information about packets. . . . 184

Chapter 1

Introduction

The topic of this thesis is the design, implementation and evaluation of adaptive traffic management for Multi User Virtual Environments (MUVEs). These are a new and valuable class of application; they offer users the ability to define their own 3D environments and to use them in innovative ways. However, the demands they make on the network are greater and more complex than similar applications like computer games. In addition most MUVE traffic does not use TCP or its congestion control and existing MUVEs have at best primitive congestion control mechanisms. Whilst current MUVEs manage to operate reasonably within today's Internet there is significant room for improvement in their use of the network. When a network is congested MUVEs can be too aggressive and conversely there are circumstances when they do not utilise their share of network resource. The work described in this thesis identifies and addresses these shortcomings.

The design of an adaptive traffic management system that meets the needs of MUVEs whilst allowing them to interact fairly with existing network traffic is described in this thesis. At the core of this design is a client that measures network conditions, calculates a fair rate and communicates requests to a server. This client is compatible with the current Second Life (SL) and OpenSim server implementations and demonstrates measurement based adaptive traffic management for today's MUVEs. This work has been motivated by an investigation into and analysis of MUVE network traffic, which develops our understanding of how MUVE network traffic fits into the mix of Internet traffic and how its needs relate to the needs of other network traffic.

This introductory chapter introduces the topic and provides context for the rest of the thesis. This chapter is organised into a number of sections. In the first section MUVEs are defined and an overview of the work is presented. In subsequent sections key aspects of the work are highlighted and the structure of the document is presented. The structure of the rest of the chapter is as follows:

Section 1.1: An overview of MUVEs and the work is presented.

Section 1.2: The goals of this work are described.

Section 1.3: The thesis statement is given.

Section 1.4: A road map of this thesis, describing the other chapters.

1.1 Overview

In this section the defining features of MUVEs [131, 115, 92, 62, 3, 117] are considered and an overview of the work in the dissertation presented.

There are a number of features which define the characteristics of the relatively new class of applications referred to as MUVEs. They are multi user systems which use a computer network to facilitate communication. They provide computer based simulated environments where multiple users interact through the proxy of avatars. Systems are usually pervasive, the world lasts for a long time and users will join and leave. This is usually supported with an architecture structure where there is a centralised server or server farm that performs the simulation. The rules of the virtual environment are usually similar to the real world, in that the laws of physics are emulated in some way. MUVE refers to environments that differ from multi-player games in that there is no rigidly defined set of goals. Instead the users are free to define and follow their own objectives. Users experience the world from the first person

perspective (usually from immediately behind) through an avatar. Typically there will be facilities for a high level of avatar customization. Furthermore, users are able not only to interact with the game environment but to actually create and shape it.

There are a number of different MUVEs that were created for different purposes. They differ in the kind of environments that they provide and the number and type of users that they support. The data model used also differs between the different systems [6, 176].

The defining characteristics of MUVEs have consequences for the network traffic that supports communication between clients and between a client and MUVE server. The network properties of MUVEs differ from those of other network applications [6, 97, 102, 109].

These characteristics will be examined in more detail later in the document. For now we note that as real-time systems MUVEs have soft real time constraints, which make use of the Transmission Control Protocol (TCP) [160] inappropriate for most of their traffic [102]. Consequently most use the User Datagram Protocol (UDP) [156] as their primary transport layer protocol. Secondly, the high level of customization of Avatars and the user control over the environment mean that the amount of network traffic required is significantly more than many games such as First Person Shooters (FPSs) [106][1][2][3][4][5][6] like Half Life [7][8] and Massively multiplayer online role-playing games (MMORPGs) [24][9][10] like World of Warcraft [185, 186][11] and others [12][13][14].

The number of high bandwidth applications being used on the Internet has been increasing recently, for example IPTV [191]. Much of the existing traffic as well as much of projected new traffic uses TCP as its transport level protocol and as a result it is congestion controlled [4]. However, a significant proportion uses UDP, which has no provision for congestion control, consequently congestion control for non TCP traffic has been and will continue to be an issue of some importance for the Internet.

When considering traffic management more generally and congestion control in particular it is necessary to consider, how the needs of the application can be met, how best the user's experience can be served and how this can be balanced with the responsibilities which come with utilizing a shared communal resource; the network. This work does so with respect to MUVEs. It is widely accepted that non TCP traffic should in some way be "TCP friendly", we explore what TCP Friendly means in this context, to what extent current MUVE traffic is TCP Friendly and where deficiencies are found how they can be addressed. Furthermore we consider the nature of MUVE traffic, and decompose it into its constituent functional components. We address both how a MUVE can obtain and maximize the utility from its fair share of network resource.

As MUVEs simulate an environment that is similar to the real world, that users can become immersed in, it is important that the system reacts to users input quickly and consistently. If there is a long delay between a user performing an action and getting a response the immersiveness of the environment will be undermined. This means that MUVEs have soft real time constraints.

A MUVE with an adaptive traffic management system will be able to change the amount of traffic that it sends out onto the network depending on the amount of other traffic and congestion on the network. In a congested network the round trip time will increase, which will adversely affect the experience of using a MUVE. A traffic management system created specifically for MUVEs will be better able to react to its needs, including being able to prioritise things

[1] Bytonic Software. Jake 2. `http://bytonic.de/html/jake2.html`, 05 May 2006. [Online; accessed 22-October-2010]

[2] Epic Games and Digital Extremes. Unreal Tournament 2004. Atari `http://www.unrealtournament2003.com/ut2004/`, 16 March 2004. [Online; accessed 28-March-2011]

[3] id Software. Quake 2. `http://www.idsoftware.com/games/quake/quake2/`, 9 December 1997. [Online; accessed 29-September-2010]

[4] id Software. Quake 3 arena. `http://www.quake3arena.com/`, 2 December 1999. [Online; accessed 29-September-2010]

[5] idSoftware. Quake. GT Interactive `http://www.idsoftware.com/games/quake/quake/`, 22 June 1996. [Online; accessed 22-October-2010]

[6] RavenSoftware. Quake 4, 2005. `http://www.idsoftware.com/games/quake/quake4/` [Online; accessed 30-June-2011]

[7] Valve Corporation. Half-life. Sierra Studios, Electronic Arts and `http://store.steampowered.com/app/70/`, 19 November 1998. [Online; accessed 07-October-2010]

[8] Valve Software. Counter-Strike: A Counter-Terrorism Half-Life Modification. `http://www.counter-strike.net/`, 5 July 1995. [Online; accessed 03-October-2010].

[9] ShenZhou Online `http://www.ewsoft.com.tw/` [Online; accessed 03-October-2010]

[10] NCsoft. Lineage II `http://www.lineage2.com/`, 1 October 2003. [Online; accessed 03-October-2010]

[11] Blizzard Entertainment. World of Warcraft [Computer game]. `http://us.battle.net/wow/en/`, 23 November 2004. [Online; accessed 04-April-2011]

[12] The Creative Assembly. Medieval: Total war. Activision, 20 August 2002.

[13] Beautiful Game Studios. Championship Manager 2008 `http://www.championshipmanager.co.uk/gamesarchive/championship-manager-2008` [Online; accessed 14-October-2010]

[14] Firaxis Games and Westlake Interactive. Sid Meier's Civilization III. Infogrames `http://www.civ3.com/`, 30 October 2001



(a) Second Life (b) OpenSim

Figure 1.1: Second Life and OpenSim screen shots.

other than maximum throughput. In traditional 3D multiplayer environments, such as games [15][16][17][18][19][20][21][22][23][24] the environment is largely unchanging and known to both the server and the client, before the session begins. This allows the network traffic to consist primarily of communicating the actions of the players of the game. The server and clients can use these to update their models of the environment.

Many MMORPGs have long sessions where users join and leave during the session. These games usually do not allow the users to make any permanent changes to the environment. FPSs often allow the users to create new maps in which to play, this is done using external tools [125][25][26][27] and the map must be created and distributed to the players before the session begins. In the non-gaming virtual environments like Second Life on the other hand the world is not known to the client before the user joins the session as the world can be changed by the users from inside the environment.

Two screen shots one, of Second Life and one of OpenSim, are shown in figure 1.1. Both of these are using the same Second Life client, in both cases the avatar is walking forward. The user controls the avatar with a computer keyboard, the forward button makes the avatar walk forward, the back button makes the avatar move back towards the camera. The left and right buttons make the avatar turn to there left and right. The page up makes the avatar jump if it is standing if press for a short period on time and if pressed longer the avatar flies up. The page down button makes the avatar fly down and stop flying if it reaches the ground.

As little about the contents of the world is known to the client before it connects to the server running the world it must receive most of the information about the world whilst connected. In this thesis it is shown that this results in the bandwidth requirement for such applications being much larger than those of a FPS or MMORPG. Such a world is not usually limited in size or number of simultaneous users, to support this the world is usually split up into a number of different servers. So long as the users do not gather in a small number of places, dividing the world into sections will work. This is the technique used by Second Life and a number of others. This means that as the user

[15] Valve Software. Counter-Strike: A Counter-Terrorism Half-Life Modification. `http://www.counter-strike.net/`, 5 July 1995. [Online; accessed 03-October-2010].

[16] RavenSoftware. Quake 4, 2005. `http://www.idsoftware.com/games/quake/quake4/` [Online; accessed 30-June-2011]

[17] Jeffrey Jacobson. Gates of Horus. `http://publicvr.org/html/pro_gates.html`, 2008. [Online; accessed 22-October-2010]

[18] idSoftware. Quake. GT Interactive `http://www.idsoftware.com/games/quake/quake/`, 22 June 1996. [Online; accessed 22-October-2010]

[19] Epic Games and Digital Extremes. Unreal Tournament 2004. Atari `http://www.unrealtournament2003.com/ut2004/`, 16 March 2004. [Online; accessed 28-March-2011]

[20] Valve Corporation. Half-life. Sierra Studios, Electronic Arts and `http://store.steampowered.com/app/70/`, 19 November 1998. [Online; accessed 07-October-2010]

[21] ShenZhou Online `http://www.ewsoft.com.tw/` [Online; accessed 03-October-2010]

[22] id Software. Quake 2. `http://www.idsoftware.com/games/quake/quake2/`, 9 December 1997. [Online; accessed 29-September-2010]

[23] id Software. Quake 3 arena. `http://www.quake3arena.com/`, 2 December 1999. [Online; accessed 29-September-2010]

[24] NCsoft. Lineage II `http://www.lineage2.com/`, 1 October 2003. [Online; accessed 03-October-2010]

[25] Epic Games and Digital Extremes. UnrealEd. `http://udn.epicgames.com/`, 2004. [Online; accessed 22-October-2010]

[26] idSoftware. Gtkradiant. `http://www.qeradiant.com/cgi-bin/trac.cgi`, 2007. [Online; accessed 08-October-2010]

[27] Raphal Quinet and Brendon Wyber. Doom editing utilities. `http://www.montefiore.ulg.ac.be/~quinet/games/DEU/DEU-en.html`, 26 January 1994. [Online; accessed 08-October-2010]

moves between areas in the game the client has to communicate with more than one of the servers.

There are a number of different MUVEs that were created for different purposes. They differ in the kind of environments that they provide and the number and type of users that they support. The data model used also differs between the different systems. Systems where the world lasts for a long time but users will join and leave usually use an architecture structure where there is a server that performs the simulation.

The adaptive traffic management system design uses a modified client allowing it to connect to the existing Second Life servers and OpenSim servers. The client based system calculates a rate as that is what the server will accept from the client to limit its sending rate. To compete fairly with traffic on the Internet rate calculation system should be TCP fair [59]. The design is implemented as a modified Second Life client called Mongoose.

In summary: the requirements of a Multi User Virtual Environment differ from those of traditional data applications, such as HTTP, FTP and email, in that they do not require every packet to be delivered. They are resilient against low levels of packet loss. At the same time they have similarities to applications like video conferencing in that they are inelastic applications and are sensitive to delay. Consequently for much of the data transmitted by a MUVE TCP is not an appropriate transport protocol. In this MUVEs are similar to multi player games, whether they be first person shooters like Quake [28] or massively multi player online games like World of Warcraft [29]. However, there is an important difference, the bandwidth requirements of MUVEs are greater than comparable classes of games. Therefore it is important for the stability of the network that they contain effective traffic management and congestion control facilities.

1.2 Goals of the work

The main aim of this work is to develop adaptive traffic management for MUVEs, that will allow the application to make efficient and fair use of available network resources. In order to achieve this aim a number of subsidiary goals have to be met, so that the design can be based upon both an understanding of the state of the art in MUVE traffic management and upon a sound understanding of both how MUVEs work and the way that their network traffic changes depending upon both network conditions and user activities. Furthermore in order to validate the design an implementation is required. This leads to the following goals:

1. Through analysis of related work to contribute to the understanding of Multi User Virtual Environment's traffic, placing MUVEs into context.
2. Measurement and experimentation to contribute to the understanding of Multi User Virtual Environment's traffic.
3. To design adaptive congestion control for MUVE which:
   (a) Adapts to changes in network conditions at time-scales appropriate for application utility and network stability.
   (b) Competes fairly with TCP and other Internet traffic and is therefore able to receive an appropriate share of network resources.
4. To implement adaptive congestion control system in a modified Second Life client, allowing it to be used with OpenSim and Second Life servers.
5. To evaluate the implementation and the design, by comparing with standard SL behaviour and standard TCP fair behaviour.

1.3 Thesis Statement

The thesis of this work is that measurement based adaptive congestion control will allow MUVE traffic to coexist fairly with established Internet traffic whilst improving the service the application receives from the network. Such a system can be achieved through:

[28] idSoftware. Quake. GT Interactive `http://www.idsoftware.com/games/quake/quake/`, 22 June 1996. [Online; accessed 22-October-2010]

[29] Blizzard Entertainment. World of Warcraft [Computer game]. `http://us.battle.net/wow/en/`, 23 November 2004. [Online; accessed 04-April-2011]

1. *A measurement system limited to the client side and therefore usable with the current Second Life and OpenSim servers.*

2. *A window tracking rate algorithm to allow timely adaptation to network conditions.*

When compared to current MUVE traffic management the system will be TCP fair over a wider range of network conditions. When compared with equation based algorithms systems, such as TCP friendly rate control (TFRC), it will allow quicker adaptation to changes in network conditions.

1.4 Road Map of thesis

This dissertation starts by establishing the context of our discussion of MUVEs by considering; the Internet, its protocols and the problem of congestion. It then introduces MUVEs, giving their properties and how they relate to other systems. The network properties of MUVEs are then described followed by a description of how other network traffic relates to MUVEs. Second Life is then analysed followed by an investigation of its properties and the properties that it requires from a congestion control system. This document then presents a congestion control algorithm that has all of the properties that are required. The evaluation of these systems is then described followed by the conclusions and future work.

The rest of the thesis has the following Chapters:

2 Internet Traffic

This chapter provides context for the work on MUVEs by looking at different traffic types and congestion control approaches. It looks at traffic management and the Internet. It describes the Internet including the IP and TCP protocols and the problem of congestion. The concept of congestion control is described along with different approaches to methods of congestion control. The congestion control implemented in TCP is described. The challenges that arise for non TCP applications, are identified.

3 Multi User Virtual Environments

This chapter characterises the properties of MUVEs. It describes other work on MUVEs and applications that are similar to MUVEs. It then gives a history of MUVEs and other systems that are similar to or predecessors of MUVEs. The properties that define MUVEs are identified and analysed with respect to the properties of other similar systems. For example MUVEs allow user control over objectives which distinguishes them from games where there are objectives or a concept of progress.

This chapter then describes Second Life. It reports what Second Life provides to the user, how it works and how it interacts with the network. A substantial and new analysis of how Second Life manages its network traffic, which is based upon analysis of source code and examination of network traffic is presented. The properties necessary for adaptive traffic management are described.

4 Methodology

This chapter outlines the top level methodology used in the work. In conducting this study; first MUVEs were analysed with respect to their network properties, next a measurement and experimental study of Second Life was conducted, the outputs of the analysis and design phases were used to inform the design, implementation and evaluation of traffic management mechanisms for MUVEs.

To support this work substantial development of experimental infrastructure was undertaken. The methods used to capture network traffic is described. A user space implementation of DCCP that was created is described. SLparse which was created to examine the traces of Second Life traffic that were captured, this program also facilitates the examination of DCCP traffic connections and the test networks utilised. The netem system which was used to change network conditions to perform experiments is described.

5 Second Life Evaluation

This chapter reports the evaluation of traffic captured from Second Life, under different network conditions. The related work on Second Life is described. The traffic was captured from a real world situation, as well as from different scenarios that were created to investigate the behaviour of the system under different situations. This chapter then gives the requirements that Second Life has on a congestion control system which have been

determined from the traffic evaluation and the reasons why an adaptive congestion control system would benefit Second Life. The different types of traffic that make up Second Life traffic are described and the different levels QoS that they need.

6 Adaptive traffic management for MUVEs

An adaptive traffic management system for Second Life is described in this chapter. It compares the TCP and the throttle adoption system in Second Life. It analyses the different algorithms that could be integrated into Second Life and how Second Life could better control its traffic. The structure of the Second Life client is described and the changes that have been made to the client to integrate the new congestion control system. The new client is called Mongoose.

7 Evaluation

An evaluation of the systems is presents in this chapter. The methodology employed in this evaluation is described. The fairness of the calculated throttle values against other traffic and different network conditions is tested. The window tracking congestion control used by the Mongoose client is compared other types of congestion control. The bandwidth utilization is compared to the requested bandwidth. The TCP fairness of the produced bandwidth is evaluated.

8 Traffic Management with OpenSim

This chapter presents an investigation of OpenSim traffic with the standard Second Life client and the Mongoose client. This chapter describes the throttle system of OpenSim and a modification that has been made to the throttle system in response this evaluation.

9 Summary and Conclusion

In this chapter it is shown that the thesis statement is innovative, significant and has been demonstrated to be true. It summarises the work in earlier chapters and their conclusions.

1.5 Summary

This chapter introduces the topic of the thesis and describes the rest of the document. MUVEs are virtual environments that use the network to allow multiple users to be in world at the same time. MUVEs are different from multi player games, consequently their network requirements are different. Specifically they use more bandwidth and have different types of traffic with different Quality of Service requirements.

The goal of this work is to understand MUVEs and to create a traffic management system that satisfies their traffic requirements. This system must do so by reacting to network conditions and be fair to other traffic on the network. The contributions of this work are a better understanding of Second Life's traffic and its existing non TCP fair traffic management system. This work also contributes an evaluation of the different types of congestion control that could be used to manage MUVE traffic and the design, implementation and evaluation of a client based system for Second Life.

The rest of the document first describes the Internet and its traffic. This is followed by a description of MUVE and Second Life in particular, then a description of the methodology employed in this work. An evaluation of Second Life traffic is presented followed by the design of a client based adaptive traffic management system. The evaluation of the described system is presented. This is followed by an evaluation OpenSim and evaluating the Mongoose client with OpenSim. A design of modified OpenSim throttle system is designed and evaluated.

Chapter 2

Internet traffic and Congestion Control

This chapter provides context for our discussion of Multi User Virtual Environment (MUVE) traffic management, by looking at current Internet traffic and congestion control. The mechanisms that make the Internet work are discussed and the challenges that arise when introducing new applications into the Internet are identified. The properties that an application needs to have in order to coexist fairly with the other traffic on the Internet are described.

MUVEs are distributed applications, they use network protocols to communicate and normally use the Internet for this communication. The most common transport layer on the Internet is the Transmission Control Protocol (TCP), over 80% [61, 89] of the data on the Internet is carried by it. Its congestion control system shapes most of the traffic on the Internet and prevents congestion collapse. The services that TCP provides to applications make it inappropriate for MUVE traffic. As MUVEs use the Internet but not TCP, they can't use TCP's congestion control, so need to control their own traffic. It is therefore important to understand the properties of the Internet and TCP congestion control to allow MUVEs to make efficient and fair use of the Internet. The remainder of this chapter has the following sections:

Section 2.1: The Internet and the challenge of congestion are described.

Section 2.2: The concept, methods and types of congestion control are described.

Section 2.2.3: The congestion control system implemented in TCP is described.

Section 2.3: Different traffic types are described and contrasted.

2.1 Internet

The network layer of the Internet is a packet switched best effort network. The Internet Protocol (IP) is a datagram, connectionless protocol, every packet contains a source and destination address. The routers don't know about any transport layer connections that exist and don't normally reserve network capacity for connections or hosts. There is no guarantee that all of the packets for a connection will take the same route. [1] There is no reliability provided by the IP layer, if a packet is lost the IP layer will not retransmit it.

The Internet is a global datagram network, it provides an unreliable service for transferring packets of data. A layered model is used where the layer n provides services to layer n+1. There are different layering models for the Internet, RFC1122 [17] gives a four layer model; Application, Transport, Internet, Link. This is illustrated in table 2.1. The main Internet layer protocol is IPv4 [159], which replaced earlier versions [155]. The latest protocol, IPv6 [42] is intended as a replacement for IPv4, among other things it addresses the problem of IP address starvation. Support for addressing is provided by IP. There are a variety of different transport types, including TCP [160], UDP [156], SCTP [179], RDP [147] and DCCP [99] which can run over IP networks. The IP datagrams can be transported by a number of different link layer protocols including Ethernet [161, 120, 37, 77], ATM [108, 38] and PPP [175]. This

[1] There has been significant research into adding Quality of Service (QoS) provision to the Internet, however this has not seen wide spread deployment. Integrated Service (IntServ) [18] allows bandwidth to be reserved and Quality of Service (QoS) to be guaranteed, the resources can be reserved using Resource reservation protocol (RSVP) [19]. Differentiated services [137, 14] (Diffserv) allows coarser-grained mechanisms for giving different service to different traffic.

Layer	Example
Application	HTTP · FTP · SSH · Games
Transport	TCP · UDP · DCCP · SCTP
Internet	IP
Link	PPP · ATM · Ethernet

Table 2.1: The 4 layer model.

design also allows IP datagrams to be encapsulated by non link layer protocols including IP and transport layer packets [15, 150, 180, 5][2].

When a host using IP has a datagram to send it will send it directly to the destination or to a router [7, 134]. The router will then forward the packet to another host, destination or another router. The method of determining the Link layer address is link layer specific. The routers have a queue of packets waiting to be sent. A router can receive a packet from one link layer protocol and send it out on a different link layer protocol. This allows for a network made up of different link layer networks. The router contains tables that indicate where a packet should be sent.

IP packets contain a field indicating which transport layer protocol they contain. There are a large number of different transport layer protocols providing different services. Some of the protocols carried by IP packets are considered Internet layer protocols as they are used for signalling at the Internet layer. Internet Control Message Protocol (ICMP) [157, 158, 124, 16] is used for error reporting, diagnostics and routing purposes.

There are competing interests in the design and construction of the Internet. How these competing interests shape systems is described in [33].

2.1.1 Transmission Control Protocol

Most of the traffic on the Internet uses TCP [160, 45] as its transport layer protocol. TCP provides a reliable, in order, duplex, byte stream abstraction to the application, which is congestion controlled and flow controlled. TCP is an end-to-end system. End-to-end design is discussed in [171]. TCP is good for applications that require reliable point to point communication without strict time constraints. As TCP provides in order delivery, if a packet is lost it must be successfully re-sent before any data after it is available to the application. For applications where latency is important, and reliability is not essential, User Datagram Protocol (UDP) [156] has been commonly used. UDP does not provide reliability or congestion control, so packets that are lost are not retransmitted. Data is delivered to the application as it arrives, consequently from the application's perspective there may be gaps in the sequence space and data may have been reordered. The fact that the application does not have to wait for retransmissions may reduce delay and enable applications to meet their timing constraints.

Thus it may make sense for some application not to use TCP. As UDP has no prevision for congestion control the application can send more traffic than their fair share and may not react properly to congestion. For non TCP applications it is often the responsibility of the application to implement their own congestion control system. Unfortunately this is a significant challenge which is often not completed satisfactorily [100].

A TCP connection starts with a three way handshake. The sequence number system used by TCP has a sequence number for each byte so when packets are lost their bytes can be re-sent. A TCP connection closes with each side sending a packet with the FIN flag set. Each FIN is usually acknowledged. To prevent a large number of small packets from being sent Nagle's algorithm [133] is used. This works by allowing only one packet that is smaller than the Maximum Segment Size (MSS) to be unacknowledged within a TCP connection. This results in TCP connections having mostly having packets of the maximum size carrying data and packets of the minimum size carrying no data with few packets in between these two extremes. TCP and UDP have port numbers which are used to allow multiple connections to be uniquely identified between two hosts. This allows multiple processes to be communicating between a pair of hosts. A connection is identified by a 4-tuple, the IP addresses of the two ends of the connection and the port numbers used at both ends.

A TCP connection starts with the server end of the connection waiting with a socket bound to a port and in the listen state. In order to open the connection that client must know the IP address and port number where the server program is listening.

[2]Florian Heinz and Julien Oster. DNS tunnel. `http://sourceforge.net/projects/nstx/`. [Online; accessed 29-March-2011]

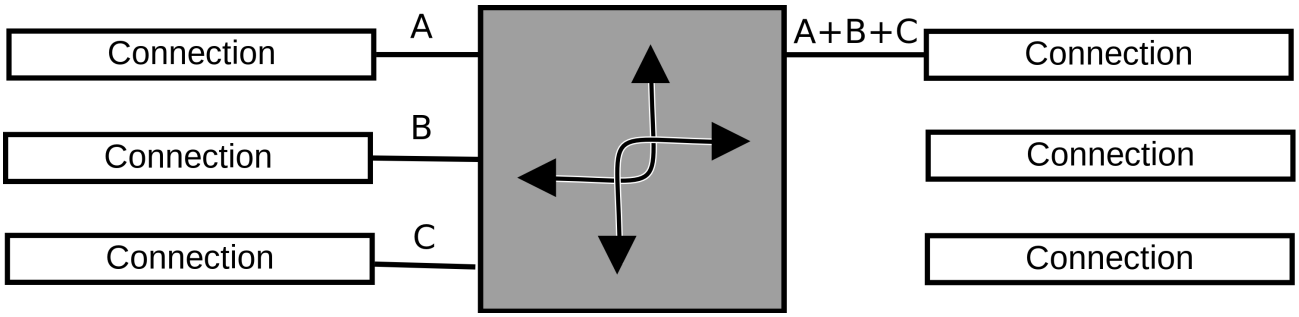


Figure 2.1: A router where congestion is happening.

2.1.2 Congestion

There is a limit to the capacity of network links, they can only carry a limited number of bytes per second. On a network there can be situations where there are several high bandwidth links and a low bandwidth link, this can cause there to be not enough bandwidth to send out the received packets. It is also possible that the amount of traffic addressed for a link will not match amount of bandwidth provided by the links coming out of a router, this is illustrated in figure 2.1. A router sends and receives data on two or more network links, that may have different amounts of bandwidth. In a router with first in first out queues and a drop tail policy, the newly arriving packets are dropped when there are too many. When a router receives packets they are queued then processed to determine their destination and then put into another queue for the outgoing link. When there is not much traffic the queues will stay small, as the packets can be transmitted soon after being queued. As the number of packets being sent onto the network increases a point will be reached when router queues will start to increase. This leads to the time taken for a packet to reach its destination to grow [87]. The amount of throughput the network achieves increases as the load on the network increases. When the load on the network nears the capacity of the network the throughput increases more slowly and the RTT increases more quickly. When the load increases so that the routers would have to transmit data faster than they are capable they will have to drop packets from the queues, resulting in packet loss. The processing capacity of routers is also finite so it is possible for packets to need to be discarded before they can be processed. When the load presented to an uncontrolled network is too high network resources are wasted. For example if a packet is dropped the resource used to transport the packet to the point where it is discarded are wasted. As a consequence after a certain point as load is increased the useful work that the network achieves is reduced. The retransmission mechanism of TCP can increase congestion on networks as the lost packets are retransmitted meaning that it will amplify the network congestion and can lead to congestion collapse.

One metric that is used to quantify the work that a network achieves is power. The power of a network is a measure of the throughput and delay. The power is proportional to the throughput and inversely proportional to the RTT. The power of the network is at its maximum when the load on the network is equal the amount of bandwidth that the network can handle [164].

2.2 Congestion Control

Transmission of data on a computer network can be regulated by two complimentary mechanisms flow control and congestion control. Flow control limits the rate of data transmission in order to prevent the receiver from being overwhelmed by the amount of data. Congestion control controls the transmission of traffic to regulate use of network resources which are shared by multiple end points. Congestion can be controlled by allocating bandwidth to connections and by detecting congestion and adjusting transmission rate.

Jain describes why the problem of congestion can't be tackled by simply improving the network in [87]. This paper also looks at the ways that congestion can be controlled. The first type of system described increases the amount of available resources to meet the demand, this works by holding back resources until they are needed. The second type of system manages resources by decreasing the demand. The second type of system is discussed in more detail. There are three methods for reducing demand:

1. The system denies the creation of a new session, if the available resources are insufficient to meet its demands.

2. When a new session is to be created the existing sessions reduce their resource utilization allowing new sessions to get sufficient resources from the network.

3. The system schedules when each session gets access to resources.

The Internet does not have sessions or a session control system. Methods 1 and 3 require a session control system that is aware of the number of sessions and control the creation of sessions. Something close to method 2 can be implemented using a congestion control system, where the clients control the amount of network resources. In order for congestion to be controlled the system needs to be aware of congestion which indicates that too much network resources are being used. There are different ways that congestion can be detected. End points may detect the loss of packets [85] observe changes in RTT [20] or receive explicit indications of congestion from routers [163]. Routers themselves can detect congestion by measuring for example queue length or inter packet arrival time they may send explicit notification [163] to end points or may simply discard packets [56].

2.2.1 Window and rate based congestion control

There are different types of system that control the amount of traffic that is sent out on to the network. It can be controlled by ether limiting the number of packets or bits in flight on the network at one time or by limiting the number of packets or bits sent on to the network per unit time. A system controls the amount of in flight packets will do this using a window and a count of the number of packets in flight. There may be multiple windows controlling the sending of data, with a congestion window reacting to congestion signals from the network, this type of system is referred to as window based congestion control. A system that limits the amount of data sent out is called a rate based congestion control system.

The window or rate used will change depending on the network conditions. A system can react to congestion events and successful deliveries by increasing and decreasing the sending of data. Alternatively a system can put the information about the network conditions into a formula to produce a value that determines the sending of data. For the formula based systems moving averages for values are usually used.

A system that changes its window value, as a result of congestion signals from the network rather than using a formula to calculate the window, is called a window tracking [179, 57, 9, 81, 85]. A system that uses a formula to calculate the windows or sending rate is called formula based [59, 58, 72]. The window or sending rates can be in bits or they can be in packets. When the control value is in bits the packet size is normally included in the formula. There are different values for the packet size that can be used by systems. The maximum segment size is often used, or a long term average size can be used. It is also possible to use a value supplied by the application or a moving average [99].

The RTT affects the rate of packet transmission when a window based congestion control system is used. This is because as the RTT increases packets remain in flight for longer and therefore the throughput decreases. Rate based systems on the other hand are not affected directly by the RTT so they do not automatically adjust to increase in RTT. Rate based congestion control usually uses a formula based system to determine the sending rate with the RTT included in that equation. Window based congestion control usually use a window tracking system that does not include the RTT in this calculation.

The functions used to determine the amount the window is increased or decreased by a window tracking system are important in how they act and interact. For bit based system the packet size is usually included in one or both of these formulas.

2.2.2 TCP congestion control

The congestion control mechanism located within TCP are the Internet's main defence against congestion. With the deployment of high bandwidth Local Area Networks (LANs), primarily Ethernet, and the continued constrained wide area bandwidth, in the late 1980's the Internet began to suffer from congestion collapse [133]. Overloaded Internet links would drop packets. The reliable delivery of data means that when packets are lost they are retransmitted. This would increase the load on the network. This positive feedback loop could result in the situation where little productive work could be carried out by the network.

> *In October of '86, the Internet had the first of what became a series of 'congestion collapses'. During this period, the data throughput from LBL to UC Berkeley (sites separated by 400 yards and two IMP hops) dropped from 32 Kbps to 40 bps.* [81]

This phenomenon stimulated research into Congestion Control for datagram networks like the Internet [81, 31, 86]. This was necessary as the original design priorities of the Defence Advanced Research Projects Agency (DARPA) Internet [31] prioritised resilience against external threat and paid little attention to traffic management. Whilst there was a Source Quench [157] option to ICMP it was optional and not implemented by many routers. Congestion control was added to the Internet in order to allow it to keep functioning.

Raj Jain defined a notion of Max Min fairness [86], where any session is entitled to as much network use as is any other unless some sessions can use more without hurting others. He designed a class of Additive Increase Multiplicative Decrease (AIMD) [164] algorithms which when receiving a signal for the presence or absence of congestion from the network result in a Max Min fair allocation of resources and operate the network at an optimal load.

In 1986 Jain [85] described a timeout-based scheme for window flow controlled networks which detects congestion and aims to prevent congestion collapse. With this system when a packet is transmitted a timer is started, if the packet has not been acknowledged before the timeout the packet is considered to have been lost, this is taken to indicate that the network is congested. A variable called the window size (WS) limits the number of packets that are concurrently unacknowledged. When a packet is lost the WS is reduced to one. For every N successful packet transmissions the WS is increased by one, up to the window specified by the other direction. This paper, advocates the use of a congestion control system where the window size is reduced to one when a packet is lost, because when packet loss occurs the number of network and router queues are all full and will take some time to empty.

Congestion happens when there are too many packets being sent to the routers. The routers can therefore detect when the network is congested, by for example monitoring queue length. This allows a build up of congestion to be detected before packet loss occurs and for the network to be operated at optimum load. Ramakrishnan and Jain in [165, 164] describe a binary feedback congestion control system where packets are marked to indicate if the router they travelled across was congested. This information is then transmitted back to the sender in the next packet. When more than half of the packets indicate that the network is congested the congestion window is decreased by 1/8th and when the network is not congested the window is increased by one segment for each window of data. This is an AIMD algorithm.

Explicit Congestion Notification (ECN) [163] is a system where Random Early Drop (RED) [56] routers mark packets instead of dropping them. It is similar to but differs from Jain's scheme in that packets are marked randomly and so different streams will receive different congestion feedback. Consequently, a statistical version of Max Min fairness is achieved where each stream has an equal chance of obtaining network resource. If there is an ECN capable transport layer protocol being used it sets the Type of Service bits to indicate this, the routers then change these bits instead of dropping the packet. The transport protocol's congestion control system reacts to the marked packet as if it had been detected as lost [179, 99, 163]. There are also Weighted random early detection (WRED) [50] routers which allow different flows to be given different priority. ECN is based on work in non IP networks and currently experimental and not deployed on much of the network.

In 1988 Jacobson [81] described a congestion control system. In this scheme the sender maintains a congestion window. The actual window size used by a connection is the minimum of the congestion and flow control windows. The design aims to allow a connection to reach a steady state of packet transmissions, where the network conditions are stable.

To get to this state the connection starts in the Slow Start phase where for each acknowledgement received the congestion window is increased by one segment. This results in an exponential increase of the congestion window for each window of data. This prevents a connection from dumping a full flow control window of packets onto the network at the start, but allows it to rapidly probe the network for resources. The connection remains in the Slow Start phase until the congestion window is greater-than or equal to the Slow Start Threshold (SSTHRESH), the connection then enters a Congestion Avoidance stage. In this stage the window size is increased by one segment for each window of acknowledgements.

As TCP already needed to detect the loss of packets in order to provide a reliable service, and as the main cause of packet loss was congestion, packet loss was taken as a convenient implicit indication of congestion. The round trip time (RTT) for the connection is estimated using an exponentially weighted average. This is combined with a measurement of the variability of RTT to calculate a Retransmit Time Out (RTO).

When packets loss is detected by the source a variable SSTHRESH is set to half the current congestion window and the congestion window is set to one segment size. Thus the TCP connection will enter a new Slow Start phase which will continue until the congestion window reaches the new SSTHRESH value, at which point the Congestion Avoidance algorithm will take over. This work formed the basis of the congestion control that was introduced to TCP and upon which the Internet still relies.

Packet loss in TCP may also be detected through duplicate acknowledgements. As TCP acknowledges the next expected byte, if there is a gap in the sequence space, subsequent packets arriving will generate acknowledgements for the first byte in that gap. If three duplicate acknowledgements are received TCP assumes that packet loss has occurred. If packet loss was detected by a retransmission timeout then the congestion window is set to one segment. If loss was detected by duplicate acknowledgements then the congestion window is halved and TCP remains in the congestion avoidance state. In both cases the SSTHRESH is set to half the congestion window [84].

The combination of Congestion Avoidance and loss detection through duplicate packets results can be considered a form of AIMD algorithm. The use of loss as a signal of congestion means that it cannot be expected that each flow across a congested link will receive the same feedback. Consequently Max Min fairness is not achieved. However a form of statistical fairness [4] is achieved, where each flow as similar opportunity to achieve access to network resources.

The addition of congestion control to TCP coincided with a significant decrease in the experience of congestion collapse on the Internet and has contributed greatly to the usability of the network. However there was and continues to be a significant and increasing volume of traffic that does not use TCP and for which it would be inappropriate to simply reproduce the TCP congestion control algorithms. It is however considered to be important that such traffic is not greedy and is responsive to the presence of congestion.

TCP is a window based window tracking congestion control system. The window is in bytes. There are several different versions of the TCP congestion control system the original [4]. [3]

To summarise, TCP's congestion control mechanisms come out of research conducted in the late eighties. Raj Jain defined a notion of Max Min fairness [86], and designed a class of AIMD [164] algorithms which result in a Max Min fair allocation of resources. Van Jacobsen designed Slow Start and Congestion Avoidance algorithms, which were incorporated into TCP [81]. Comparing Jain's [164] and Van Jacobsen's work [81], they differ in that Jain advocated the modification of routers to provide accurate feedback to hosts. Van Jacobson advocated using loss as a signal of congestion thereby allowing congestion control mechanisms to be deployed with a relatively small change to TCP's source code. The combination of Congestion Avoidance and loss detection through duplicate packets is an AIMD algorithm, which achieves statistical fairness [4], each flow has similar opportunity to obtain network resource.

2.2.3 Modelling TCP Congestion Control

It is possible to characterise the behaviour of TCP. The window size that it uses in its Congestion Avoidance steady state is inversely proportional to the square route of the rate of packet loss. The macroscopic or steady state behaviour

[3]

Tahoe: Treats 3 duplicate acknowledgements as a timeout.

Vegas: Vegas uses RTT as an indication of congestion and calculates the throughput for the current RTT and window and for the RTT when the link is not congested, using the minimum value seen for this. The difference between these two sending rates is calculated, when it is too large the window is decreased and when it is too small the window is increased.

Binary Increase Congestion control (BIC): BIC [190] is congestion control system for TCP that is designed to better utilize bandwidth whilst competing fairly with the different varieties of TCP and itself. The increase and decrease functions that it uses varies depending on the state of the connection. When the window is below a certain value it acts like TCP, when the window is above this value it uses a binary search to find the correct sending rate. It also attempts to compete fairly when different connections have different RTT. BIC converges faster than Reno though there are some concerns about its stability.

CUBIC: CUBIC [169] is an enhanced version of BIC that uses a different function for the increase of the congestion window. This simplifies the system as it does not now have several modes. CUBIC converges faster than Reno but slower than BIC.

The formula for different TCP can be given as:

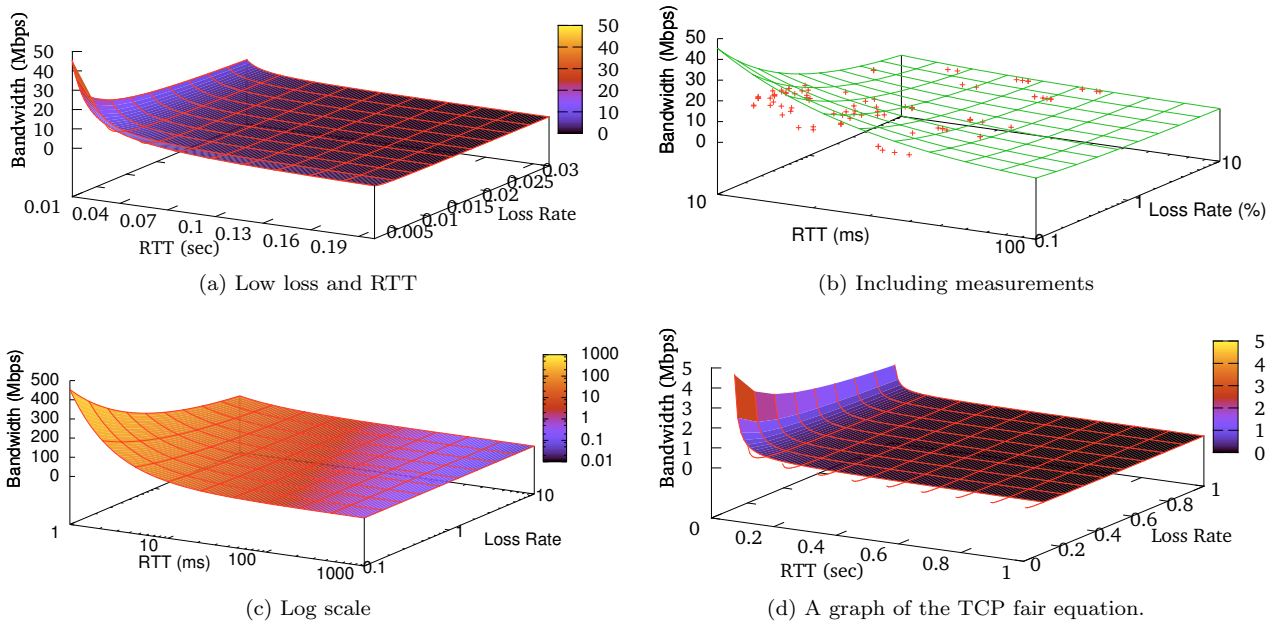
$$X = \frac{c \cdot MMS}{RTT \cdot p^d} \tag{2.1}$$

Where c and d are constants. For standard TCP $d = 0.5$. This equation given in [190]

$$\left(\frac{RTT_2}{RTT_1}\right)^{\frac{1}{1-d}} \tag{2.2}$$

for the ratio of bandwidth utilization for two connections sharing a link, where the d is the same as in equation 2.1. It also states the values of d are between 0.5 and 1 different TCPs have different values for d which means that they have different RTT unfairness.

Scalable TCP (STCP): STCP [93] is designed to provide much higher throughput and scalability. The long term bandwidth allocation formula for it has a d value of 1 (from [169]) meaning that it is unable to compete fairly at different RTTs. STCP uses a different increase and decrease functions where the CWND is increased by 0.01 of its value per ACK and set to $0.875 * CWND$ when a loss event is detected. This means that the time between peaks in bandwidth usage is dependant on the RTT and not the sending rate.



(a) Low loss and RTT

(b) Including measurements

(c) Log scale

(d) A graph of the TCP fair equation.

Figure 2.2: Four different views of the TCP fair equation over a range of RTTs and loss regimes.

of TCP can be represented by the TCP Fair equation [118] below 2.3. [4]

$$X = \frac{S}{RTT\sqrt{p}} \tag{2.3}$$

Graphs of the TCP fair equation are shown in figure 2.2. The x-axis shows the RTT in seconds, the y-axis gives the loss rate and the z-axis gives the bandwidth the equation allows. The surface is coloured to represent the z-axis value.

(a) The TCP fair equation for loss rates between 0.5% and 3% and RTT between 10 ms and 200 ms is shown in figure 2.2a. At higher levels of congestion and at higher round trip times the throughput achieved by TCP is considerably less.

(b) Average values from real traces along with the surface from the TCP fair equation are shown in figure 2.2b. The data points are illustrative of the relationship of the equation to the behaviour of TCP. The experiments these traces are from are described further in chapter 6. The equation surface is not coloured to make it possible to see the points. The majority of the points are at or below the equation line, this shows that the equation does give a good estimate of the amount of bandwidth used by TCP for low bandwidth. The points are below the line for low loss and RTT because TCP is also limited by the flow control window.

(c) A log scale of the TCP fair equation for loss rate between 0.1% and 10% and 1 ms and 1000 ms is show in figure 2.2c. The x and y axis are log scale as well as colouring of the surface. This is the log scale version, to expand the interesting part.

(d) A 3D plot of the TCP fair equation for up to 100% loss and 1 second RTT is show in figure 2.2d. This is over a larger range than the other graphs and is rotated.

This equation characterises the steady state behaviour of TCP for long lived connections. They provide a benchmark against which the throughput utilised by non TCP connections can be compared. It shows that for environments where congestion is low high throughputs of multiple megabits per second can be expected from TCP.

The formula gives information about the congestion window of TCP connections, however TCP connections are also affected by a number of other factors.

[4] TCP Friendly Rate Control [72] uses a more complex formula [146]

- Flow Control Window: Connections are also constrained by the flow control window. The flow control window indicates the amount of data that one end of the connection can accept from the other, it is not affected by congestion or other network conditions and is not intended to deal with congestion [160].
- Size of Connection: Connections that never reach their fair share of the bandwidth or their flow control window, as a result of being short lived and not involving much data, may experience no loss and so be limited in their bandwidth utilization only by the round trip time [93].
- Convergence: The conditions on computer networks change with time, both on a long time scale and on a short time scale. It is important for a congestion control system to be able to adapt to changing network conditions. When the conditions change it takes time for the connections to reach their steady state [190].

  Not all of the congestion control systems that have TCP fair long term packet transmission rate also respond to congestion at the same rate and by the same amount. Congestion control systems that have the same formula for their long term congestion control and compete fairly with TCP under the correct conditions are looked at by Bansal and Balakrishnan in [9]. When routers use Random Early Drop (RED) [56] this system competes fairly with TCP, however when other policies are used for dropping packets at the routers these congestion window evolution formulae are not guaranteed to converge with those used by TCP. These congestion control systems will also converge when competing with TCP or any of this family of systems.
- Packet Size: The IP and TCP headers are each a minimum 20 bytes each, so the size of the packets used will affect the amount of overhead from the packet headers. TCP normally waits until it has several bytes [133] before it sends a packet rather than sending a byte as soon as there is data to send, network resource are available to send it and the client buffer space ready to accept it. If the mechanism is left enabled it can reduce the data sent out on to the network without reducing the amount of user data transported, however it will reduce the responsiveness of the system because it introduces extra delay.

  When a router receives a packet it has to, at least, calculate the next hop that the packet should be sent to and copy the IP level packet into a new network level packet for that next hop. The time taken copying and encapsulation and de-encapsulation may be affected by the size of packet, the amount of time to look up the destination will however not be affected by the size of the packets as it does not involve inspecting the payload of the packet [174, 11].

  When a router is unable to process or forward all of the packets it is receiving it will have to drop some of them, this is network congestion. The number of packets and the number of bytes sent out onto the network both affect the level of congestion. A network where all of the packets are of the maximum size for that network will carry more data than a network with smaller packets. This is a problem for some multi media applications as they need to transmit frames at a constant rate, so would prefer to be able changes the amount of data that they send but not the rate at which they send it the out. The TCP fair equation can be given in a form where it gives a rate at which packets can be sent rather than the rate at which bytes are transmitted [72].

TCP Friendly Rate Control [72] uses a more complex formula [146] that models the specific behaviour of TCP RENO and takes into account such factors as flow control window and multiple Slow Start phases. The performance of TCP for networks with a high bandwidth delay product are looked at in [104].

Figure 2.2b shows values for the TCP fair equation taken from TCP connections on a test network, it also shows the TCP fair equation. These show that there is a correspondence between the actually behaviour of TCP connections and the TCP Fair equation. At low levels of loss the throughput achieved is considerably less. This is accounted for by the limiting factor of TCP's flow control window [160].

In Summary, congestion control is important to the stability of the Internet. The congestion control introduced into TCP was effective in preventing congestion collapse. However there is a significant proportion of traffic for which TCP is not appropriate as a protocol. These applications should use some form of congestion control which responds to congestion in ways similar to TCP. TCP's behaviour can be characterised by the TCP fair equation which provides a useful point of comparison against which non TCP applications can behave themselves. Much MUVE traffic utilises UDP rather than TCP consequently when looking at the behaviour of MUVE traffic it is relevant to compare the load they present to the network, and their responsiveness to changes in network conditions with that of TCP.

2.2.4 Other Congestion Controlled Transport Protocols

There are other congestion controlled transport layer protocols.

Datagram Congestion Control Protocol (DCCP)

DCCP is a transport level protocol that provides congestion control, without reliability. In [55] the problem statement for DCCP is described. As well as providing congestion control DCCP is intended to work with network address translators (NAT) and firewalls, this is possible as it is a connection oriented protocol. [99] is the RFC that describes DCCP.

Currently DCCP has two congestion control systems. The default is TCP like congestion control [57] that provides a congestion control system similar to the one used by TCP, which is described in [4]. This allows applications to receive as much bandwidth as possible, whilst having similar behaviour to TCP's congestion control.

The other congestion control system is TCP-Friendly Rate Control (TFRC) for DCCP [58], which provides a smoother rate of packet transmissions whilst remaining TCP friendly, at the expense of not achieving the maximum possible speed. TFRC is described in [72].

Stream Control Transmission Protocol (SCTP)

SCTP [179] like DCCP is connection oriented and provides packet rather the stream semantics. It provides both ordered and unordered deliver of data. SCTP is ordinarily reliable though there exists an extension to send packets unreliably. A congestion window is used to control the sending data, it implements a congestion control system similar to TCP's.

2.2.5 Summary

To prevent or control congestion on a network it is necessary to control the amount of bandwidth used by the applications. This can be done in a number of ways, the design of the Internet means that to use any type of open-loop access control system to manage the connections or the amount of bandwidth that they can use would require changes to the network, for example IntServ [18].

The methods of control data transmission are to control the rate of data transmission and a window system to control the number of in flight packets or bytes. The value for these systems can be calculated in two ways, using a formula to calculate a value from moving averages of the connection statistics, maintaining a window which is adjusted in response to the arrival of packets and detection of loss.

TCP uses a window tracking system to calculate a window value to control the number of in flight bytes. The window tracking system used by TCP is AIMD, its congestion detection system is based on detection of loss. The AIMD window tracking system used by TCP converges to an even share for different connections under the same conditions. This system requires loss to happen and be detected to work, the window based systems move around the optimal value. TCP allows multiple connections to share a network. This arrangement leads to suboptimal sharing of resources.

2.3 Internet Applications and Traffic Types

For applications with real time constraints TCP is often not appropriate, consequently the question of how non TCP applications should regulate their behaviour is an ongoing one. The TCP Fair equations have been used as a reference point for evaluating their behaviour [9, 8].

Low bandwidth applications such as Internet chat will only use a small proportion of the bandwidth that is available on a network. There are a number of applications which are bursty, they will use a large amount of bandwidth for a short period of time for example when File Transfer Protocol (FTP) is used to download a DVD quality or better movie. Different types of real-time multi media applications use different amounts of bandwidth, the amount they use can sometimes be adjusted to match network conditions. Applications that do not have strict timing requirements and want as much bandwidth as possible use TCP. TCP's congestion control system [4] is designed to react quickly to changes in the available bandwidth and to use as much as possible of the available bandwidth, whilst providing statistical fairness between the connections. TCP fairness means that similar connections that are not limited in packet transmission by the application will all get an equal opportunity to share the available bandwidth. The rapid changes in the amount of available bandwidth are not a problem for applications trying to move large amounts of data around without any strict timing constraints. This type of congestion control cannot provide a steady transfer rate when network conditions are not constant. Various types of multimedia system, require that their data is transmitted at a predictable rate. Whilst they usually are able to adapt changes in the available bandwidth, they must still keep

to their timing constraints, this can include having to transmit packets at a regular rate and so the decrease in the use of bandwidth may involve sending small packets rather than packets of the same size less often. For video and audio streaming changes in packet size can often be achieved by changing the encoding, this does not usually result in a system that can exactly match the amount of bandwidth that is available to it or that can change the amount of bandwidth that it needs at the packet granularity.

To avoid congestion such applications employ a number of techniques. Some systems limit their traffic to set level or attempt to reduce the amount of traffic that is to be sent over the network so that there is no need for congestion control. They may alternatively use a congestion control system at a level above the transport level.

- Telnet. [154] Telnet is an application that has been used on the Internet for a long time. It is a text based protocol that does not normally carry large amounts of data. It provides the user with a command prompt access to another machine, the keystrokes are sent to the other machine and the results sent back, so like MUVEs it is important the reply is sent back promptly.

- Hypertext Transfer Protocol (HTTP). [54] The still images and text, that make up most of the traffic carried by HTTP, take up little bandwidth, these are elastic applications in that there is not a deadline by which data has to arrive. However, increases in delay will decrease the quality of the user experiences and significantly so if the delay is in the order of several second or more [170]. The importance of this delay is less than in MUVEs also the total transfer time is more important than the instantaneous transfer rate during the connection.

  Large files transferred using HTTP and FTP can consume a large amount of bandwidth. The total time taken from when the transfer starts and when it ends is important for these types of application. The stability of the transfer rate is not important to bulk transfer systems, only the long term bandwidth.

- Peer-to-Peer. Peer-to-Peer applications [91, 90] can also be used to transfer large files or large numbers of files this can consume even larger amounts of bandwidth as a large number of connections are often used by peer-to-peer applications. Many peer-to-peer applications use TCP so their bandwidth usage is controlled by TCP's congestion control system, however each TCP connection used by such applications are separate with regards to congestion control. The traffic of peer-to-peer applications will therefore get more bandwidth than the traffic of other applications that use less connections. Peer-to-peer systems can also be used for purposes other than bulk transfer [176].

- Video and Audio conferencing: Video conferencing [69, 80] allows two way communication, it requires a small delay. It is possible to create video conferencing systems with different levels of bandwidth requirements depending on the quality of video that is required and the amount of bandwidth that is available. Packet loss can be handled through a form of redundancy or by displaying blank data for the missing parts.

- Video and Audio streaming: Video and audio streaming [76] has also become more popular, the requirements of these applications are also high bandwidth and timing maybe of some importance though the constraints are weaker than applications that contain an interactive element such as real-time interactive applications such as Video conferencing. When video or audio is streamed the variation in the time taken for packets to cross the network is important. With non-interactive video streaming the time taken for packets to cross the network

Traffic Type	**Connections**	**Bandwidth**	**Time Constraints**
TELNET	One	Very Small	Within seconds
FTP	Two or more	Large	Usually unimportant
Web Traffic	Variable	Small	Elastic
Peer-To-Peer	Large number	Large	Usually unimportant
Video Streaming	One or Two	Medium to Large	Elastic
Video Conferencing	One to many	Medium	Real time
Audio Streaming	One	Medium	Elastic
Telephony	One	Medium	Real Time
Games	Usually Less than 20	Small	Soft Real Time
MUVE	Many	Medium	Soft Real Time

Table 2.2: Type of network traffic.

is not very important, though with live transmissions it may be important that there is not too large a delay between events happening and them being seen. It is possible to create video streaming systems with different levels of bandwidth requirements depending on the quality of video that is required and the amount of bandwidth that is available. Packet loss can be handled through a form of redundancy or by displaying blank data for the missing parts. Unlike MUVEs streaming applications will have little or no data that has to be sent reliably.

- 3D Interactive Games: The other new application area that has become more popular recently is that of 3D interactive games with which 3D virtual environments have much in common. The First Person Shooter (FPS) Counter-Strike has an average bandwidth utilization of 40 kbps [52] and the Massively multiplayer online role-playing game (MMORPG) ShenZhou average 7 kbps [24]. The timing for different types of network games differ depending on the level of interactivity of the game. FPSs require a small RTT and a small variation in that RTT. MMORPGs however can deal with longer RTTs as it does not involve fast paced action.

  In a traditional 3D multiplayer environment, such as a game the environment is unchanging and known to both the server and the client, usually before the session begins. This allows the network traffic to consist of the actions of the players of the game and any non-player characters in the environment. The server and clients can then share a model of the environment. Many MMORPGs have long sessions where users join and leave during the session. These games however usually do not allow the users to make any permanent changes to the environment. FPS often allow the users to create new maps in which to play, this is done using external tools and the map must be created and distributed to the players before the session begins. In the non-gaming virtual environments like Second Life on the other hand the world is not known to the client before the user joins the session as the world can be changed by the users from inside the environment.

In applications which have timing constraints packets that are lost often should not be re-sent as they may miss their play-out deadline by the time they arrive. Most of these applications can use techniques such as forward error correction to smooth over gaps in the data stream. Table 2.2 summarizes the different traffic types.

The requirements of MUVEs are different from those of the other types of application. They differ from Internet data applications such as HTTP, FTP and Peer to Peer file sharing in two respects, firstly they require a relatively long lived stream of data, secondly they are less elastic.

They differ from Audio and Video streaming in that there is more room for differentiation within a MUVE data stream than a video or audio stream. If avatar control packets are delayed this will have a more noticeable effect than some other background traffic. If an audio sample is delayed it will have the same effect as if other audio samples are delayed.

Like games virtual environments are highly interactive and so have timing requirements that are stricter than web traffic. They produce far more traffic than network games. MUVEs differ from MMORPGs in that they have higher bandwidth requirements and that MUVEs have a wider variety of traffic types each with a different impact on user utility.

2.4 Summary

The Internet is a packet switching best effort network, which uses IP packets to transport datagrams. IP packets are transported by the different link level protocols that make up the Internet. The addresses in the IP packets are not inherently related to the addressing system that is used by the link level protocols. The routers connect together networks with different systems below the IP layer.

The majority of the traffic on the Internet uses TCP as its transport layer protocol. TCP provides an in order reliable and congestion controlled byte stream abstraction to applications that use it. Congestion is caused when there are more packets sent onto the network that it can handle, this causes packets to be lost. A reliable protocol that resends can contribute to congestion collapse where the network is doing little or no useful work.

Congestion control involves reacting to indications of congestion to prevent congestion collapse. The congestion control systems used also attempt to allow each congestion controlled connection to get a fair share of the available traffic. There are different types of congestion control, they either control the number of packet, or bytes, per unit time or they control the number of in flight packet. The congestion control value can be calculated by maintaining a window that increase or decreases in reaction to congestion and non congestion or the value can be calculated from history of the connection.

TCP uses a window tracking window based congestion control system. It uses packet loss as an indication of congestion and uses AIMD. There are a number of different types of TCP. The majority of traffic on the Internet uses TCP as its transport layer protocol, so is controlled by TCP's congestion control system. The applications not using TCP should therefore use a congestion control system that competes fairly with TCP.

There are different types of application traffic on the Internet, that have different requirements on the network. Multi User Virtual Environments have properties that are different from existing types of application.

- TCP provides the congestion control of the Internet as it is used by most but not all of the applications that use the Internet. There are different types of application that have different requirements from the transport protocol, some of which are not compatible with TCP, so they do not use TCP. There is therefore a responsibility for congestion control to be implemented for theses applications, MUVEs are applications of this type.
- TCP Congestion Control has been effective therefore it is an important point of reference for congestion control systems. It is however not optimal and there are other types of system, for example Jain's work on binary feedback for congestion control.
- Different application have different requirements. There is a need for a concrete analysis of MUVEs to understand their needs.

Chapter 3

Methodology

The methodology used in this dissertation is to perform an analysis of Multi User Virtual Environments (MUVEs) focusing on Second Life in particular, then perform a measurement study of Second Life. The information extracted from this was analysed and used to inform system design. The designed system was then implemented and then evaluated with Second Life and OpenSim.

This chapter describes the process that was involved in investigating the relationship between Transmission Control Protocol [160] (TCP) and Second Life. The tools used to investigate Second Life traffic are also described.

The measurement and investigation of Second Life and OpenSim traffic is described. This involves passive measurement of the traffic between the client and the server. The source code of the Second Life client was investigated along with the source code of the OpenSim server. This information was analysed to develop an understanding of Second Life Protocol (SLP) traffic.

A system to make Second Life traffic TCP fair was designed, this required that the properties of TCP fair traffic be understood. The Second Life client was investigated and a method of integrating a TCP fair throttle calculation into the client was designed and implemented. This chapter describes that process along with the tools and techniques that were employed.

This chapter is arranged into the following sections:

Section 3.1: The systems used to capture packets is described.

Section 3.2: A program created to analyse packet traces is described.

Section 3.3: The NetEM system is described.

Section 3.4: The process involved in this work is described.

3.1 Packet Capture

To analysis network traffic it is necessary to monitor traffic. There are two types of monitoring active and passive. Passive monitoring is where packets on the network are captured an analysed. Active monitoring is where packets are transmitted onto the network and the response is analysed. Active monitoring can evaluate the performance of the network and network protocols, however, it is not possible to discover the performance of the protocols in use [29]. Passive monitoring also does not change the traffic on the network and can be used to discover the behaviour of network protocols in their operation. Active measurement can be used to determine information about the network including loss rates and round trip times (RTT). An example of active measurement is using ping packets [158] to measure RTT. An example of using passive measurement would be to observe a packet and its acknowledgement. The information investigated in this work is the behaviour of Second Life traffic in real networks.

This section describes the mechanisms used to capture network traffic and the libraries and storage formats used for examining and storing captured traffic. Capturing and storing packets allows them to be analysed after the session has finished. This allows the analysis program to have more flexibility in the process, so it could be slower than real time or take multiple passes over the trace.

The BSD Packet Filter (BPF) [119] is a packet capture system that includes a filtering system to reduce the number of packets that are transferred to user space. The filtering can be performed on the packet without copying it, so that

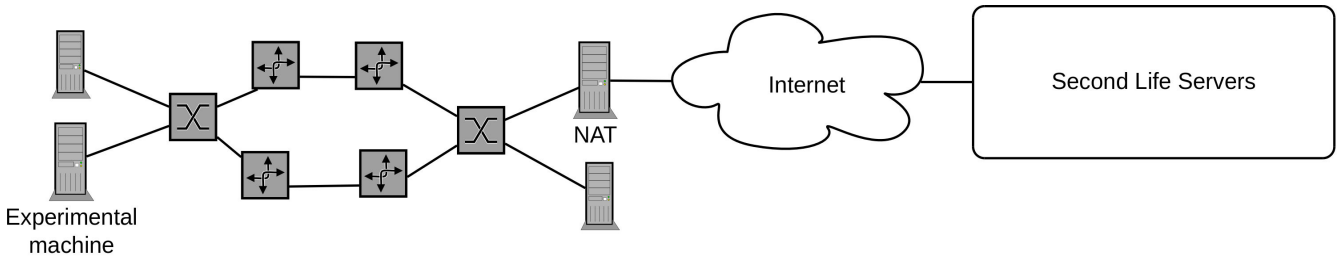


Figure 3.1: The structure of the test network.

discarded packets are never copied. BPF uses acyclic control flow graphs to examine the packet and decides if it is to be accepted or rejected. The pseudo-machine used is designed to work well on a system with a number of registers and relatively slow memory access. It is not specific to any particular protocol so it does not need to be updated for new protocols. This filtering system is implemented in several packet capture systems. The original implementation [119] as well as a number of others are run in kernel memory without first copying the packet.

CMU/Stanford. Packet Filter (CSPF) [123] uses a stack based interpreted language to filter packets. The intended use of the packet filter is to demultiplex packets, the use of the filter network monitoring is also suggested.

Mach Packet Filter (MPF) [192] is a packet filter system which is implemented as an extension of BPF. MPF was first implemented as part of the Mach 3.0 Operating System, as Mach [2] is a microkernel architecture the networking code is not in kernel so MPF was designed to allow multiple filters to pass the correct packet to the correct protocol servers. The design is such that every process that uses, for example TCP has a complete TCP stack in its address space, there could potentially be a large number of filters in memory. MPF combines similar filters into a single filter.

The PF_Packet protocol family can be used on Linux systems to get all of the packets that are accepted by the network interface, by using this with a card put into promiscuous mode this can be used to capture all of the traffic on a network. A BPF filter can be attached to the socket so that only the desired packets are received.

Routers can also perform packet capture, though they will usually take a sample of the packets rather than capturing all of the packets that they receive. The captured information is then put into a format such as Netflow [30] or sflow [152] and sent somewhere else for analysis.

libpcap [83] is a library used by a number of systems that perform packet capture as well as a number of user space network stacks. It provides an interface for capturing packets from a wide variety of network devices. It provides a system independent interface for capturing packets from the network and for examining traces. It does not directly perform packet capture, it uses a number of Application programming interfaces (APIs) including BPF and PF_PACKET to perform packet capture. It uses BPF filters to filter the traffic according to input. It also includes a BPF compiler to produce BPF programs from understandable statements. The packets read can be processed by the program using the library or they can be output to a file which can be later read by libpcap and processed like packets from a network.

The tcpdump[1] program is used in this work to capture packets, it uses libpcap to capture packets. It dumps the packets into the libpcap packet dump format. This captured traffic can be analysed using a separate program which uses libpcap, this program is called SLparse.

3.2 Analysis program: SLparse

A tool was created to analyse DCCP connections [2] and Second Life circuits, this tool was written in C [94]. It was created as a single program so that it could also be used to analyse Second Life circuits transported by DCCP packets. The program can also analyse DCCP packets that are encapsulated inside UDP packets. This tool detects different connections by their source and destination addresses. The program generates graphs for information extracted from the trace. TCP connections are also detected and their packets grouped to allow the graphs to include information about TCP connections.

The number of packets and the number of bytes is counted for different intervals. The intervals used are 10 milliseconds, 100 milliseconds, 1 second and 1 minute. The number of packets and bytes a second is calculated from

[1] V. Jacobson, C. Leres, S. McCanne, et al. Tcpdump. `http://www.tcpdump.org/`, 1989. [Online; accessed 03-November-2010]

[2] DCCP is a protocol designed for datagram traffic like Second Life and OpenSim. We have developed a user space implementation and have demonstrated it working with a modified OpenSim server and client. This is documented in appendix B

this. The numbers of bytes and seconds for the different protocols are also counted and output for the different time intervals. The supported protocols are TCP, UDP, DCCP and Second Life circuits. The number of application levels bytes is calculated for different units of time overall and per protocol. The maximum number bytes per second and the maximum number of application bytes per second is also calculated. The distribution of packet sizes is recorded and output as the Probability Distribution Frequency (PDF) and the Cumulative Distribution Frequency (CDF). The frequency distribution the number of bytes a second is recorded and output as the PDF.

The number of bytes and packets per second is also recorded for each connection or flow. The data is output in the correct format for generating graphs using gnuplot [3]. The per connection data is output separately for each connection and so that it can be drawn on a single graph.

The program outputs a textual representation of the trace, that is similar to tcpdump, for the simple information about packets. The information about the state of the connection is also included in the output.

3.2.1 Related work

There are several other programs for analysing traffic that are similar to SLparse that are described below, however none of them have the desired features.

CoMo [78] is a system for generating statistics about network traffic. It has input plug-ins that read packets either from the network or from files containing already captured traffic. The analysis is done by modules which receive every packet, along with a piece of memory to hold information about the flow to which the packet belongs. As CoMo is designed for generating statistics of values relative to time the flow objects are stored away at each sampling interval. The system allows a large variety of modules to be created. The modules cannot be chained together so two modules that perform similar tasks must each perform their task, separately.

CoralReef [95] is a system that can be used to perform packet capture and analysis. It contains two software stacks for parsing network traffic, one that parses packets and one that handles flows. It can read packets from a variety of sources including libpcap and its files and its own file format. It is implemented as a library, which can be used to create network monitoring programs. It can use libpcap to use BPF filters on the traffic that it reads in.

Bro [149][4] is an Intrusion Detection System (IDS) which uses port numbers [167, 168, 189] and pattern matching [36] to identify user level protocols it then runs the packet through modules that track the state of application and transport level sessions to detect suspect behaviour. Bro uses libpcap to capture packets and BPF filters to take only the packets that it needs from the network. The packets are then passed to a system that examines connection behaviour protocols and hosts to look for anomalies behaviour and then generate events. The events can cause a number of things to happen, including generating logs and running scripts.

Wireshark[5] is a system that can perform live capture of traffic and analysis of traffic as well as the offline analysis of captured traffic. It can identify protocols and capture files being transferred over the network as well as perform analysis of application level protocols. Wireshark is capable of reading and writing a number of different packet capture formats including the libpcap format.

The structure and operation of Nprobe is described in [127]. Nprobe [126] has modules to process and analyse the IP and TCP headers as well as the modules to process application level protocols. The paper describes the Hypertext Transfer Protocol (HTTP) and Hyper Text Markup Language (HTML) modules that can examine HTTP connections and extract the transmitted files. The HTML module can then parse the HTML to extract information on browsing patterns and references to other objects. This information can then be used to create web load generators to test and compare web servers. This data can also be used to determine unnecessary overheads in the web browsers and servers. This paper also states that Nprobe has more modules and can be used for a variety of other purposes. A visualisation tool constructed as part of Nprobe is described in [70].

Nprobe has also been used to study the behaviour and performance of web browsers [71]. The transferred HTML is examined to identify the HTTP and Domain Name System (DNS) connections that are part of the page. This allows the performance of the web browsers to be examined.

Snort [6] is an IDS which uses libpcap to capture traffic. It uses pattern matching and protocol analysers to detect and block unwanted use, or misuse, of a network.

[3] T. Williams and C. Kelley. GNUplot: an interactive plotting program. `http://www.gnuplot.info/` 1998.
[4] Vern Paxson. Bro. `http://bro-ids.org/` [Online; accessed 29-September-2010].
[5] Gerald Combs. Wireshark. `http://www.wireshark.org/`, 1998. [Online; accessed 29-September-2010]
[6] Martin Roesch. Snort. `http://www.snort.org/`, 1998. [Online; accessed 29-September-2010]

Benko and Veres, in [12], describe a method of estimating the packet loss rate, this method ignores packets whose deliver status cannot be accurately determined to give an accurate rate of packet loss. This will only work if those packets whose deliver status can be accurately determined are representative of all of the packets.

The programs described above do not have the desired features, the only program that can recognise and parse SLP packets is Wireshark. The problem with Wireshark is that it cannot produce the desired analysis or the tables and graphs used is this thesis.

3.2.2 DCCP

For DCCP connections SLparse keeps track of the state of the connection and the features. Information about the number and size of packets is recorded and used to generate graphs. By keeping track of the features it is possible for the program to determine the congestion control system or systems that are being used by the connection. The program can then use this information to track the state of the congestion control system. Packet loss can be detected and the round trip time (RTT) calculated. DCCP uses time stamps and includes delay information in packets so the network delay can be calculated separately from the other delays. When packets are captured at the ends of the connection the sending and receiving time for the time stamp packets is known. The DCCP macro state is tracked by using an implementation of the state machine that can deal with losing some of the packets that lead to state transition. The state of the connection after the packet has been sent or received is included in the text output. The contents of the headers and the options are included in the textual output.

3.2.3 Second Life

Support was added to SLparse to allow it to track the state of Second Life[115] circuits. It groups together packets that belong to the same circuit and treats them like connections for the purposes of the functions not specific to Second Life traffic. Support is included to parse the headers of packets [153] and print this information. The packet type information is then used to group packets into the throttle groups and generate graphs from this information. Information about the size of packets for different types of packet and for different circuits are extracted by the program and along with the packet type.

3.2.4 Passive loss estimation

Second Life packets have sequence numbers which can be used to identify packet loss. The loss detection system used by Second Life is a timeout based system that operates at the receiving end of the connection and relies on detecting gaps in the sequence range. The timeout is 16 times the RTT. The system keeps track of the sequence number that it expects to see next. If this sequence number is seen then the value is incremented otherwise the sequence number is searched for in the list of sequence for potently lost packets and if it is found this sequence number is removed from the list. If it is not found and the packet is not marked as having been re-sent all of the sequence numbers between this one and the expected one are added to the list of possibly lost and the next expected sequence number is set to this packet sequence number plus one. This can also be used in the calculation of fair transmission rates [27].

Oliver et al. describe a method for estimating packet loss in TCP connections in paper [143]. The packets for each connection are first grouped together. For each connection an object keeps track of which packets have been seen and which packets have been acknowledged. When the connection starts, the initial sequence numbers, of each side, are taken from the packets with the SYN flag set, this also give the client and server ends of the connection. When packets travel across the Internet there is a chance that they will arrive in a different order than they were sent. The connection keeps track of the sequence number of the next expected in-order packet, when packets arrive in the correct order this is increased to the last sequence number of each packet as they are processed. When there are packets whose sequence numbers are greater than the expected value, the list of non-contiguous blocks of sequence numbers is examined. When the packets sequence number matches the expected value, the value is updated and the non-contiguous blocks are examined. Packet loss estimation is looked at by Sommers et al. in [178].

3.2.5 Passive RTT estimation

Second Life uses special packets for estimating the RTT of the circuit. This makes calculating the RTT straightforward. A packet with the type `StartPingCheck` is sent by the side wanting a RTT calculation, this packet contains an ID number. The other end sends back a packet of type `CompletePingCheck` in response to this packet, this packet also

has an ID number that indicates the `StartPingCheck` packet that it is in response to. When the `CompletePingCheck` packet is received the time between its arrival and the time the last ping was sent is calculated. If there is a difference between the ID of the last `StartPingCheck` sent and this packet's ID then this value is multiplied by 5 and added to the time, because pings are sent out at a constant rate. The maximum of this value the old value and the time the system has been waiting since for a reply with an ID higher than this packet is taken. This value is then multiplied by 0.8 and added to the new sample which multiplied by 0.2, this smooths the RTT as well as ensuring that the system reacts quickly to increases in RTT.

Jiang and Dovrolis, in [88], describe two methods for estimating the RTT of a connection. The first works by taking the time between the SYN and ACK of the 3-way handshake. To achieve this, the time of the first packet in the connection is recorded and the number of packets in the connection is counted. If the 3rd packet results in the connection reaching the ESTABLISHED state and the time between the first and second packets and the second and third packet are roughly equal, then the time between when the connection started and the time of the third packets is taken to be the RTT. This method results in one or zero RTT calculations per connection. For long connections the value is possibly only accurate for a small part of the connection. This technique is not entirely reliable as the handshake phase of a connection is not handled by the same code as the rest of the connection. The second method involves taking the time between the bursts of packets during the slow start phase of the connection.

The technique described in [23] improves on the accuracy and quantity of RTT estimations that were performed by other techniques. This technique is designed to closely follow the technique used by implementations of TCP as is described in [4]. The RTT is calculated by taking the time between a packet and its ACK. This gives the RTT between the monitor and the receiving host so the RTT is separately calculated for each direction, the RTTs for both directions are then added together to get the RTT for the connection. This technique works by recording the time, that each sequence number, was last received. When a packet with a valid acknowledgement number is received all of the sequence numbers, going in the opposite direction, less than the acknowledgement number are examined, to find the time between the ACK and the packet that triggered it. The record of the acknowledged packet is then removed from the list.

When packet retransmission happens TCP implementations stop calculating RTT as the relationship between packets being sent and packets being received gets less predictable. To achieve this in the connections a Boolean variable for each direction is used, this variable is set to false when a retransmission is detected. If this variable is false the RTT is not updated, but if the ACK number is larger than the largest retransmitted sequence number for that direction then the variable is set to true. If the variable is true the RTT for that direction is then updated, the difference in time of the ACK and the time taken from the list sequence numbers is calculated, if the RTT is zero it is set to the calculated value, otherwise it set to the 7/8 of the old RTT plus 1/8 of the calculated value, as is described in [148]. This can result in a greater number of RTT calculations than other systems provide if there is useful traffic in both directions. If there is no traffic in one direction a connection may produce no RTT calculations.

3.2.6 Summary

The SLparse program was created to analysis the packets that are investigated in this dissertation. It can examine DCCP, UDP, TCP and SLP. The loss of packets in the connection can be detected and RTT values can be calculated. The program can monitor the state of connections and can statistics generate statistics from information extracted from the packets.

3.3 NetEM

NetEM [73][7] provides network emulation, it is for testing protocols by emulating the properties of different network. It is a queue discipline (qdisc), which allow packets to be scheduled to be sent. The qdisc NetEM can delay packets, drop packets, duplicates packets, corrupt packets and re-order packets.

The packet delay can be set to a length of time in milliseconds, a variation in milliseconds and percentage dependence on the last packet. The distribution of delays in the range can be set. The packet loss can be set to percentage, giving probability of each packet being dropped and the dependence of each successive probability on the last. The packet duplication is specified as a percentage duplicated. The packet corruption introduces a single bit error into

[7] S. Hemminger. Netem network emulator. `http://linux-net.osdl.org/index.php/Netem`. [Online; accessed 29-September-2010]

the percentage of packets specified. The packet reordering can be either be a fixed sequence of packet were one of the group is not delayed and the rest are or it can be specified as the percentage of packets that are not delayed.

3.4 Process of this work

In this section the process used to perform this investigation is given. This gives an overview of how the different methods of enquiring were applied at each stage of the work. It also shows the reader how the tools described in previous sections were used. Nine stages key stages of this work are identified, each is enumerated below.

1. Analysis and evaluation of TCP and DCCP
2. Analysis of MUVEs and Second Life
3. Measurement study of Second Life
    - Validate previous studies
    - Real World Traffic
    - Controlled experiments
4. Design a client based traffic management system.
5. Evaluate the client based traffic management system.
6. Controlled experiments with OpenSim
7. Evaluate the client with OpenSim
8. Design a modified throttle system for OpenSim
9. Evaluate the modified throttle system.

First TCP was investigated, traces of TCP traffic were examined which helped with the understanding of the operation of TCP under real network conditions. Modules for CoMo [78] were created and these were used to analyse traffic in pcap [83] format. Traces in the text output format of tcpdump [82][8] were then processed using a program that we created for this purpose called tcpread [143]. Traffic capture is described in section 3.1.

Congestion control was looked at. The system in TCP and the theory behind different types of congestion control. The tcpread program extracts information about conditions in the trace and track congestion control values. The Datagram Congestion Control Protocol [99] (DCCP) was investigated as were different implementations of it. There was not at that time a user space implementation of DCCP that was compliant with the specification, so the User-Space DCCP Library (USDL) implantation was created. The USDL implementation is described in further detail in appendix C. The SLparse program, described in section 3.2, was originally created to help in the creation and testing of USDL, it was named after it was created.

USDL was used to test different congestion control systems. The congestion control systems that DCCP includes are CCID 2, a TCP-like congestion control, and CCID 3, an implementation of TFRC. The CCID 2 implementations of USDL and the Linux kernel were tested and determined to perform similarly to TCP's congestion control. The CCID 3 implementations of USDL and the Linux kernel were tested and determined not be able to compete successfully with TCP or CCID 2 under many network conditions.

MUVEs and Second Life in particular are of interest in academia for a number of reasons including educational [64, 63, 182, 105]. MUVEs were investigated, focusing on Second Life, this investigation focused on the network characteristics of the system. The Second Life client was examined particularly the part that calculates a rate to send to the server to control the amount of traffic the server sends to the client. The documentation on the Second Life packet types was examined to determine the function of different packets. The existence of different channels for different types of packet was determined and then the OpenSim source code was examined to determine which channel different types of packet belong to.

The structure and operation of Second Life was investigated by several methods: reading the literature examining Second Life. The operation of the Second Life client was observed. The traffic from Second Life was examined.

[8] V. Jacobson, C. Leres, S. McCanne, et al. Tcpdump. `http://www.tcpdump.org/`, 1989. [Online; accessed 03-November-2010]

The source code of the Second Life client was examined as well as the source of the OpenSim server. Finally the documentation of Second Life was examined.

The structure of the Second Life network interaction is described by Antonello et al. in [6]. The other papers [53, 6, 97, 188, 103, 102, 109] describe features of Second Life.

The SLparse program was adapted to parse SLP packets using the documentation of the packet types. The program used the information extracted from OpenSim and packet type descriptions to classify packets into different channels. The RTT calculation and loss detection systems of Second Life were determined from the source of Second Life and then implemented in SLparse.

Traffic from Second Life was captured and analysed using the SLparse program. At this time a lab session to introduce users to Second Life happened and we were able to capture the traffic between the users and the Second Life servers. This traffic was analysed and then controlled experiments were designed to further investigate the behaviour of Second Life traffic. The RTT values that the SLparse program calculated from the traces was compared to RTT values from pinging the servers that the client communicated with.

The study of the traffic generated by a lab of people using Second Life allowed us to gain insight into the behaviour of Second Life traffic in a non controlled environment. This also allowed us to determine that it is possible for a lab of user to access Second Life without problems and to determine the types of network conditions that they might encounter.

Controlled experiments were performed which were similar to previous studies, where the same action was performed for the length of the trace and the traffic characteristics were compared to these previous studies. The resulting rate were in the same range confirming that there was no significant inconsistency. The structure of the network used in the control experiments is shown in figure 3.1. The activities of Walking, Standing Flying and Teleporting were undertaken in the Second Life environment. This allowed the relationship between the in world activity and the traffic to be investigated. The avatar moved around two different islands, this allowed the traffic from different islands to be compared.

The network conditions were changed using netem. The correct operation of netem was confirmed before a session using the active measurement technique of sending packets across the network to determine there loss rate and RTT. These values were then compared with the values extracted from the traces of sessions using the network.

Different methods to make the bandwidth utilization of Second Life TCP fair were considered. A method using a modified client connecting to the existing Second Life servers was decided on. Different congestion control systems were considered for the modified client. The decision to use of the modified client required the system be rate based, insight from the evaluation of DCCP suggested that there could be problems with an equation based rate control system. Simulations of equation and window tracking rate calculation systems were performed and it was determined that window tracking would better meet the needs of Second Life. The modified client, called Mongoose, was designed and implemented. There are other Second Life clients, Hippo and Meerkat are two examples, these are modified versions of the Second Life client, they are less widely used and less reliable.

A system that involved modifying the client and server was also designed, this system uses OpenSim [131] as the server.

The Mongoose client was then evaluated. The client was put through a set of controlled experiments where the avatar walked around an island with the network conditions were changed. The client when first created had no interval between the performing the throttle calculations and it was found to not to control the amount of bandwidth used by the server as well as the Second Life client. The client was then modified to include an interval and this client was then evaluated. The client was then tested with different intervals.

The criteria for evaluating the Mongoose client are as follows. The calculated throttle values were recorded and compared with TCP fair values calculated for the network conditions. This allows the TCP fairness of the client throttle values to be confirmed under set conditions. The throttle values for different loss levels were compared. This allowed the relationship between the loss and throttle value to be compared with the TCP fair values and curve shape. The bandwidth values were compared against the throttle values to confirm that the throttle system is able to control the bandwidth. The bandwidth values were compared against the TCP fair values for different loss levels.

The Mongoose client was then evaluated with an unmodified OpenSim server. The avatar was put through different activities and then the avatar walked around an island under different network conditions. It was found that throttle system in the OpenSim server limits the bandwidth in a way that limits the overall bandwidth to less than the total throttle value. A modified OpenSim server was created and tested. The same methodology was used in evaluating the Mongoose client with the unmodified and modified OpenSim servers as was used in evaluating it with the Second Life servers.

3.5 Summary

A range of tools and techniques have been used to investigate MUVEs. These measurements of real systems have informed the design of systems that deal with shortcomings of current systems. The main feature being the creation of real systems that can be evaluated in real situations.

Passive monitoring was used in this investigation, this involves packet capture. There are a number of different system that can be used to capture traffic libpcap was used for capturing traffic and examining captured traffic. libpcap uses BPF filters to allow it to filter the packets captured. The packets captured then need to be examined. The SLparse program can examine DCCP and SLP packets and connections. It can track the state of connections and extract statistics and generate graphs. The work also involved investigating the effect of different network conditions. The NetEM queue discipline can be used to emulate different network conditions.

Chapter 4

Multi User Virtual Environments

This chapter provides context for the work by describing Multi User Virtual Environments (MUVEs). The goal of this chapter is to provide an understanding of the key characteristics of MUVEs and put them into the context of other 3D virtual worlds. A simulation of an environment that is similar to the real world, that users can become immersed in, is provided by a MUVE. The simulated virtual environments provided by MUVEs usually resemble the real world, in that it is not possible to walk through walls and things usually fall downward. MUVE usually only refers to non-gaming environments or environments that are not exclusively for gaming. The interaction between the client and server and different servers differs between the different MUVEs. The term multi user virtual environment can be taken to mean any system that provides multiple users with a virtual environment, which would include all multi player games and a large number of non games multi user systems. If the definition is restricted to 3D, multimedia environments then a lot of old games are removed and many non game systems[1], which would not have been considered MUVEs, are explicitly ruled out. The definition of MUVEs requires that they are free roaming and allow a large number of activities, Massively Multiplayer Online Role-Playing Games (MMORPGs) and first person shooters (FPSs) and other types of game types of 3D games are not therefore included in the definition as they are more restrictive and have predefined goal. All virtual environments can be used for gaming but the type that will be referred to as MUVEs in this document were not designed with gaming as there primary goal. This chapter will develop our understanding of MUVEs, there has been a lot of work on 3D games but less on MUVEs.

Second Life [115] is a popular MUVE, with 769,000 users as of 2009 [2], that we have decided to focus on because its client is open source. The properties of Second Life will be described focusing on there network properties. The avatar that the user controls is simulated at the server side as well as all of the physics. It uses UDP to transport its application level packets. It has connection like constructs called circuits, these contain sequence number that allow loss to be detected. Second Life has a system to control the bandwidth used by the server, this involves the client sending rate to the server that it uses as a limit. There is a system in the client to adjust the throttle value in response loss, this system is not TCP fair.

The remainder of this chapter has the following sections:

Section 4.1: The history of 3D multi user environments and MUVEs. Discussing the virtual environments that were created before MUVEs and the games that are similar that proceeded MUVEs.

Section 4.2: The definition of a MUVE. This section defines MUVEs by describes their properties.

Section 4.3: An examination of Second Life. This section describes Second Life and its traffic.

Section 4.4: A comparison of games and MUVEs. This section compares MUVEs with the types of multi player games that are similar to MUVEs. It also describes the literature on these types of games.

4.1 MUVE History

Multi user systems that allow interactions to occur within a virtual environment go back to the early seventies. Advent [40] was the first computer game to be based on Dungeons and Dragons. It had a text based interface and was multi-

[1] James Aspnes. Tinymud. `http://tinymud.sourceforge.net/`, August 1989. [Online; accessed 04-April-2011]

[2] `http://www.secondlife.com/whatis/economy_stats.php` [Accessed 2009]

user. Later a new version of it was created called Multi-User Dungeon (MUD) [3]. All games of this type are now referred to as a MUD. The users of a MUD moved around in a shared environment and killed dragons and other monsters. The shared environment in MUD was a maze of dungeons.

In 1989 a system called TinyMUD[4] was created, without the dragons and points. This system allowed users to create their own rooms. Systems of this type were used for a number of purposes including education [49]. They provided a virtual environment where the users could communicate in real time. These environments do not have the normal properties of games in that there are no goals or progress.

MazeWar [5] was a multi-user graphical game in which the user had a first person perspective and shot at other players. A game called Avatar involved a graphical virtual world where multiple users fight monsters. It included an economy based on gold, where there was a set amount of gold in the game. The gold started in the possession of the monsters and could be acquired by the players to buy things.

The first graphical virtual world that supported a large number of simultaneous players was called Habitat for the Commodore 64 [130, 129], it used a system called Quantum Link for communication.

In the mid-1980s commercial MMORPGs began to become available, the earliest of which were not graphical for example Adventure [40], Neverwinter Nights[6] being the first graphical MMORPG. Later MMORPGs used the Internet for their communication. Meridian 59 [7] was the first Internet based MMORPG, it included a 3D engine allowing the users to move around in a 3D world. It was limited to 250 users in the same world at the same time. Ultima Online [8] provided a top down view of the world and allowed a large number of simultaneous users.

WebWorld [22] was a 2.5D virtual world where users could chat, travel and build. Active Worlds [22] was first released in 1995. Providing a simple virtual environment. The MUVE called There [117], began closed beta testing in 2001 and was released in 2003. Second Life was also released in 2003, it provides a 3D environment with simulated physics similar to, though simpler than, the real world. Quest Atlantis [10] is a MUVE learning environment for children using Active Worlds. Croquet was released in 2007 [176, 92], it is a toolkit for creating collaborative virtual worlds. Croquet is implemented in the Squeak implementation of Smalltalk [79][9]. Open Cobalt[10] is based on Croquet, which provides a software platform for constructing virtual worlds. Edusim[11] is an educational MUVE that uses interactive whiteboards, it currently uses Open Cobalt.

Over the years there has been a progression from text based to graphical to 3D environments. Multi user environments where the users are free to define their own environments are more demanding than other game genres on computational and network resources. It is only in recent years that the network infrastructure and computational power has become widely available hence we are at an early stage in the development of 3D virtual worlds. This work is a contribution to meeting the challenges that they present.

4.2 MUVE Definition

This section will give a definition of MUVEs by identifying and describing their key features. A list of popular MUVEs will be given followed by the properties that distinguish different MUVEs. This will then be followed by a definition of MUVEs.

4.2.1 Example MUVEs

This section contains a list of MUVEs and describes their properties.

Second Life

Second Life[115] is a popular MUVE. Users can rent islands where they build things, these islands can be private. It has an economy with a currency that can be exchanged for real currencies. The 3D environment it provides is a set

[3] Roy Trubshaw and Richard Bartle. Mud1, 1978.

[4] James Aspnes. Tinymud. `http://tinymud.sourceforge.net/`, August 1989. [Online; accessed 04-April-2011]

[5] Greg Thompson. Maze war, 1973.

[6] BioWare, Neverwinter Nights, Infogrames/Atari MacSoft `http://nwn.bioware.com/` [Online; accessed 18-November-2010]

[7] A. Kirmse and C. Kirmse. Meridian 59. Near Death Studios, Inc. `http://www.meridian59.com/`, 15 December 1995. [Online; accessed 22-October-2010]

[8] Origin Systems and Electronic Arts. Ultima online, September 1997.

[9] Alan Kay, Dan Ingalls, and Adele Goldberg. Squeak Smalltalk. `http://www.squeak.org/`. [Online; accessed 29-September-2010]

[10] Community source and Duke University. Open Cobalt. `http://www.opencobalt.org/`, 2009. [Online; accessed 08-November-2010]

[11] Greenbush Education Service Center. Edusim. `http://edusim3d.com/`, September 2007. [Online; accessed 08-November-2010]

Islands, some of them joined together, between the islands is sea. It is used by several million people for a number of different proposes. It uses a client server model, where the simulation of the environment is performed on the server and the client does the visual parts. The client is open source and available for Windows [12], OS X[13] and Linux[14]. It is hosted by Linden Labs with servers at a small number of locations.

OpenSimulator

OpenSimulator [131] is a virtual world simulator server that can be used to create 3D worlds similar to Second Life. It can use the Second Life client program. The simulated worlds can be separate or connected together as a grid. OpenSim is implemented in C# [46] and runs on Mono [139] or Microsoft's .Net [121]. The source for OpenSim is available under an open source licence, making it possible to make changes to it. The network protocol between the client and servers is the same as that used by Second Life.

Active Worlds

Active Worlds [3] provides a 3D virtual environment. There are several separate worlds, some owned by Active Worlds and some owned by other people. This allows the worlds to be private or public. It was originally intended as the 3D equivalent of a 2D web browser, where users would create 3D environments instead of web sites. Active worlds is based on a client-server architecture. There is no currency in Active Worlds and it has limited physics simulation.

There

There [117] is a commercial 3D virtual environment, focusing on socializing. The users have avatars that can be customized and changed. There has a currency that can be exchanged for real currency. It is possible for users to create objects, some of this is done using external tools. Objects can be placed in the environment but cannot be created in world. There does not currently support private areas or worlds. The There client only supports recent versions of Windows.

Wonderland

Wonderland[62] is a toolkit for creating 3D collaborative virtual worlds. It is based on project Looking Glass 3D [15] and Java3D [41], and implemented in Java [116, 68]. Wonderland is still in development. Objects must be created using external tools, though it is intended to support in word creation of objects. Objects can be placed in world. A physics engine will be added in a latter release. The system is open source and can be run on platforms that support Java.

Croquet

Croquet [176, 92] is a collaborative 3D virtual environment, used for developing collaborative online multi user environments. It uses Open Graphics Library (OpenGL) [135] to do 3D rendering. Croquet is an object oriented system implemented on the Squeak virtual machine. It uses a peer-based network architecture, where messages to the shared objects are sent across its peer-to-peer network to all of the instances of the object. The computation is replicated across all of the nodes that contain that set of objects. The replicated messages are executed in a deterministic order. There is a concept of internal and external messages. External messages are messages generated by events external to the system, as a result they have to be replicated to all of the replicas of the Island. The internal messages will be generated by all of the systems locally, so do not need to be sent over the network. Time in the system only advances when messages arrive, these messages advance the current time and all messages before that time are executed in the correct order. Locally generated external messages are not directly added to the message queue but instead are sent to the Router which gives the message a time and sends them out to all of the replicas of the island state. The Router also sends empty messages in order to advance the state of the Island when there is not external input.

[12]Microsoft. Windows. `http://www.microsoft.com/windows/` [Online; accessed 08-November-2010]

[13]Apple Inc. Mac OS X `http://www.apple.com/macosx/`, 24 March 2001. [Online; accessed 08-November-2010]

[14]GNU Project, Linus Torvalds, Richard Stallman, and many others. Linux. `http://www.kernel.org/`, August 1991. [Online; accessed 08-November-2010]

[15]LG3D developers. Project Looking Glass (3D Desktop Project). Sun microsystems. `https://lg3d.dev.java.net`, 29 January 2007. [Online; accessed 04-April-2011]

Latency will cause the system to appear unresponsive as external messages from the local machine must be sent to the Router first and will be scheduled for the future from its perspective.

Verse, MetaVerse, Moove and HiPiHi

There are also other less widely used and closed source system and tools, these are described in this section. Verse [48] is a network protocol that allows 3D programs to operate as one program. It allows several programs to alter the same 3D environment, keeping it consistent for all of them.

The OpenMetaverse Project [107] was a project to create multi-participant shared virtual world. They produce the libsecondlife library, which is now called libopenmetaverse, which is used by OpenSim as well as client applications. The TextSL [60][16] client uses this library.

There is a commercial MUVE called moove Online [128]. It supports customizing avatars, listening to music with others and sharing files.

HiPiHi[17] is a MUVE that is currently in beta testing. It allows the residents to create objects and has an economy.

4.2.2 Taxonomy of MUVEs

Within the above definition of MUVEs there is room for significant diversity and there are a number of different MUVEs which illustrate this diversity. In this section we identify some of the key design choices for MUVEs, provide a summary of some of the current key MUVEs and provide a comparative table of their characteristics.

Licensing: The licensing of the client and server differs between the MUVEs. The client may be open source[115] or under a proprietary license [117]. For a system that uses a server the server code base may be proprietary or open source.

Users: MUVEs differ in the type of people that use the system, the reason that people use the system and what they use the system for [117, 115]. They also differ in the number of users, both the number that can be supported, in the system as a whole, in any given area of the virtual world, in the number of users that currently use the system and how many users use the system at anyone time.

Economy: Some MUVEs have an in world currency that can be used to buy, sell and rent goods and services[115]. This currency can also sometimes be exchanged like a real currency.

Access Model: There are different access models that are possible for MUVEs. There could be a specialized client used to access a single virtual world [117], a client used to access one of many virtual worlds [112], or a system using a web interface to access one or many virtual worlds.

Data and Computation Model: There are different data and computation models that are used, the simulation of the virtual world can happen in different places, the client[92] or the server [115]. The data model used differs between the different systems. Systems where the world lasts for a long time but users will join and leave often usually use a system where there is a server that performs the simulation.

Communication Model: There are different models for network communication for MUVEs. In a client server model the servers can be located in a single site, or a small number of sites, or the servers can be located in a number of different sites. There also exist MUVEs that use a peer-to-peer architecture. Second Life [115] employs a client, server farm model where the world is modelled in a set of servers run by Linden Labs. Others like Croquet[92], use a peer-to-peer model where the world is physically distributed between different computers. The choice of model depends not only upon technical issues such as efficiency but may also be influenced by the business plan and security issues.

Network Protocols: The network part can also differ in the transport level protocol used. TCP [160], UDP [156], Real-time Transport Protocol (RTP) [172], Datagram Congestion Control Protocol (DCCP) [99] and Stream Control Transmission Protocol (SCTP) [179] are all possible options.

[16] Eelke Folmer. TextSL. `http://textsl.org`, November 2008. [Online; accessed 15-November-2010]

[17] HiPiHi Co. Ltd. HiPiHi. `http://www.hipihi.com/index_english.html`, 19 March 2007. [Online; accessed 14-November-2010]

Traffic Management: The MUVE may have no explicit way of controlling traffic flow, simply sending packets when the application requires. If it makes use of TCP then each TCP connection will control the flow of packets that it is asked to transport. If UDP is used then some traffic control mechanism maybe built into the application which may in turn be "TCP Friendly" [72].

4.2.3 Definition

A MUVE provides a 3D virtual environment where multiple users can interact through avatars. An avatar is a representation of the user in the virtual world. Each user has their own perspective on the world this perspective is however consistent with that of other users. The provided environment is not fixed and can be altered by some or all the users in the world. In general the physics that a MUVE simulates is an approximation of real world physics, so the avatars cannot move through walls and things fall down than rather up. There are often also differences from the real world, for example avatars often have the ability to teleport and/or fly. This provides a realistic and immersive environment without the restrictions of the real world. The users' avatars will usually initially appear human but can be customized by the user.

In MUVEs users may be able to exercise significant control over the nature of the environment they operate within. This distinguishes them from First Person Shooters and Massively Multi Player Online Role Playing Games which do not allow user control of the environment. MUVEs provide a persistent environment. Consequently changes that a user makes to the environment, such as creating a building, persist from one session to the next. MUVEs are not designed specifically with a set of goals for a user. Rather the user is free to create goals of their own. This distinguishes them from gaming environments where there are a defined sets of goals which the user accomplishes to progress in, or win, the game. It is of course possible for MUVE users to think up their own goals and constructed games within MUVE environments.

Such a world is not usually limited in size or number of simultaneous users; to support this, the world is usually split up into a number of different servers. So long as the users do not gather in a small number of places, dividing the world into sections will work. This is the technique used by Second Life and a number of others. This means that as the user moves between areas in the game the client has to communicate with more than one of the servers.

MUVEs are of necessity distributed and rely upon network communication to allow distributed users to interact within the virtual environment.

This leaves us with seven central characteristics of a MUVE:

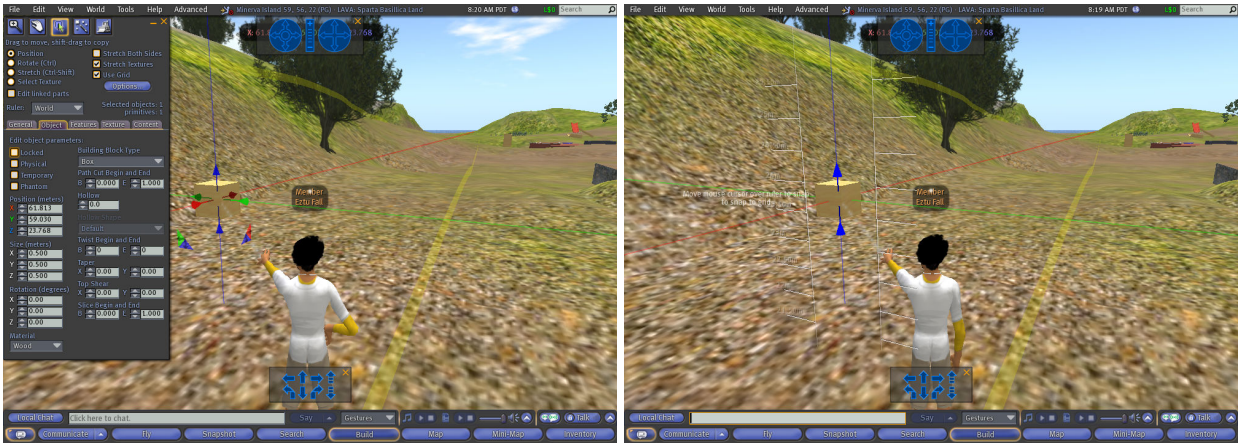
- Realistic: the environment captures essential aspects of the real world which allows navigation and communication to be intuitive.
- Persistence: the environment persists over a long period of time as do changes that are made to that environment.
- Mutability of environment: users are able to create content for and edit spaces within the environment and not simply arrange or rearrange objects provided by the system.
- User control over objectives: There are no predefined set of goals, these can be created by users of the system.
- 3D: The world is 3D providing a realistic environment
- Avatar: the world is experienced through the proxy of an Avatar.
- Distributed: MUVEs are of necessity distributed allowing humans in diverse locations to interact with each other through their avatars.

4.2.4 Summary

MUVEs are 3D virtual worlds, where the user interacts through an avatar, without any predefined goals. Different MUVEs differ on a number of properties, licensing, users, economy access model, data and computation model, communication model, network protocols and traffic management. Second Life is a commercial client server model MUVE with an economy. OpenSim is an open source server compatible with the Second Life client. Croquet is an open source peer-to-peer MUVE.

Environment	Licensing	Economy	Worlds	Data Model	Communication Model	Network protocol
Active Worlds	Proprietary	No	Many	Server simulation	public and private servers	TCP
Second Life	Open source	yes	Single	Server simulation	server farm	UDP
OpenSimulator	Open source	No	Many	Server simulation	public and private servers	UDP
Wonderland	Open source	No	Many	Server simulation	public and private servers	TCP
Croquet	Open source	No	Many	Clients	peer-to-peer	TCP
There	Proprietary	Yes	Many	Server simulation	public and private servers	TCP

Table 4.1: Comparison of MUVEs



(a) Editing the properties of the object.

(b) Moving the object.

Figure 4.1: Object creation and manipulation in Second Life.

4.3 Second Life

This work focuses on Second Life [115] which is popular having, as of 2009, 769,000 users and dominates the field. It was launched in 2003. The client program is free and open source, it is available for several different operating systems. The users, referred to as Residents, interact using avatars. It uses a client-server model of network interaction and uses UDP for most of its network communication. Connections, called circuits, are established between the client and server and between the servers. These circuits can provide reliability to some of the packets, whilst allowing others to be unreliable. Screen shots of object creation are shown in figure 4.1.

Second Life has been used for a number of different purposes in academia [98] some local examples are: A virtual archaeological excavation by Getchell et al. [64], this is based upon a real excavation of a Byzantine Spartan Basilica [187]. In this environment students are able to conduct surveys of the site in 3D, participate in a collaborative virtual excavation, construct a virtual exhibition and explore a 3D reconstruction of the Basilica.

A WiFi Virtual Laboratory (WiFiVL) in SL to support honours and masters modules in computer networking. Students are able to construct 802.11 wireless protocol scenarios by placing networking components around an Island and configuring a traffic matrix created by Sturgeon et al. [183].

Thirdly, as part of a Human Computer Interaction course students have designed and implemented tools for learning about Dijkstra's shunting algorithm [43].

Second Life can also be used for a number of other activities including music concerts [138, 1], games [114] and university lectures [18]. In addition, Second Life can be used to develop environments which mimic the behaviours

[18] Vue – Virtual University of Edinburgh `http://vue.ed.ac.uk/` `http://slurl.com/secondlife/Vue/205/53/30` [Online; accessed 01-October-2010]

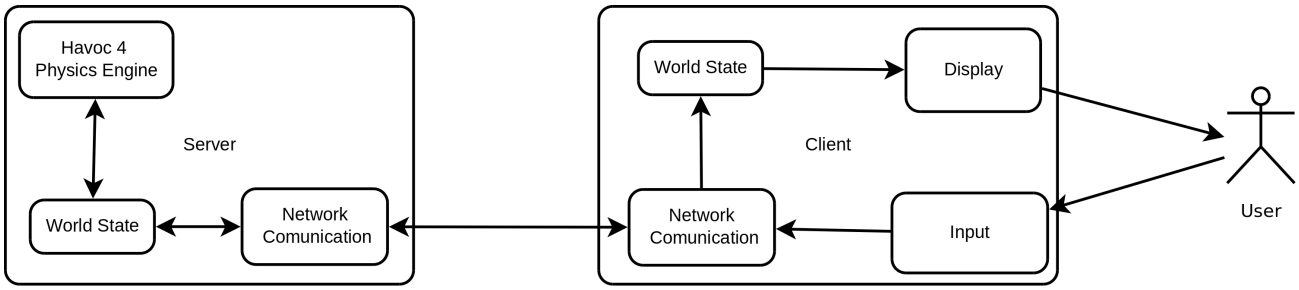


Figure 4.2: The structure of Second Life.

exhibited by *First Person Shoot'em'up* (FPS) [19], *Real Time Strategy* (RTS) [20] and *Action* type games.

4.3.1 Second Life Structure

Second Life uses a number of different servers: some that record information about the entire simulated world and other servers that simulate a part of the virtual world. Fernandes et al. examines the bandwidth utilisation of Second Life in [53]. As the environment in Second Life is dynamic, shared and the clients do not have a model of the world, all of the simulation of the environment happens at the server. The user input is sent from the client to the server, where it affects the simulation, the graphical features of the environment are sent to the client, which renders the world for the user.

The input from the user is detected at the client, this causes packets indicating the input from the user to the server. The server receives these packets puts them into the physics engine which is used as force on the avatar. The physics engine simulates physics for the island. The physics engine used by Second Life is Havoc 4. The server sends information about changes to objects, including avatars, in the environment, this gives the location, speed and acceleration of the object. This structure is show in figure 4.2.

OpenSim and Second Life use the same network protocol. OpenSim uses an implementation called libsecondlife[21]. The libsecondlife library was originally implemented to allow client programs to connect to the Second Life servers. The libsecondlife library was originally created by reverse engineering the protocol of Second Life.

In Second Life the environment is run on a collection of servers, each of which is responsible for running, typically, several Sims [6]. Each Sim corresponds to a distinct geographical space within Second Life. Currently the servers are located in a limited number of physical locations. Consequently communication between the client and server takes place across the Internet and Second Life traffic contends with other traffic for possibly scarce network resources. For example a client may be connected to servers for:

1. Login server which performs the user name and password authentication and determine the start node for a user. The location depends on the whether home, last location or a Uniform Resource Locator (URL) is specified.

2. User Server handles instant messaging sessions.

3. Space Server handles the routing of messages between locations. The Sims register their location with the space server and query it to determine the identity of the neighbouring nodes.

4. Data server gives access to the central databases.

5. Sim servers. Each Sim runs a distinct geographical space within the MUVE. Although the user's Avatar and client will be physically located on one Sim it may be able to see multiple Sims and therefore will need to be connected to their respective servers. All of the spaces simulated by the Sims are squares 256 metres in length. The Havoc[22] physics engine is used to perform the simulation [111].

[19] RavenSoftware. Quake 4, 2005. `http://www.idsoftware.com/games/quake/quake4/` [Online; accessed 30-June-2011]
[20] W. Studios. Red Alert 2, 2000.
[21] libopenmetaverse `http://www.openmetaverse.org/projects/libopenmetaverse` [Online; accessed 28-March-2011]
[22] Havok. The Havok Physics Engine. `http://www.havok.com/`, 2000. [Online; accessed 29-September-2010]

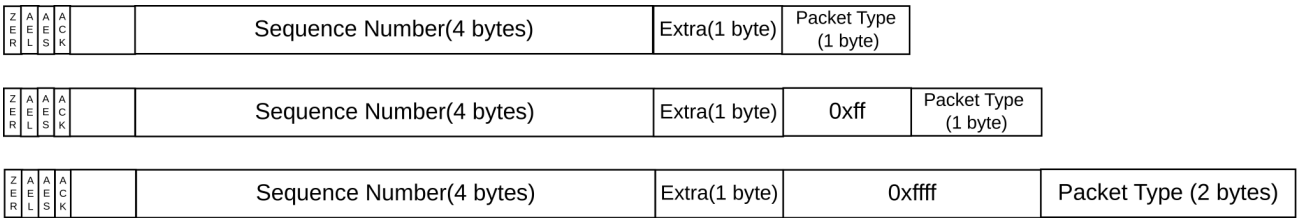


Figure 4.3: The Second Life packet header structures

4.3.2 Circuits

A circuit is a connection, established on top of UDP, which facilitates communication between a client and a server or between servers. Packets in a circuit have sequence numbers, this is known from the examination of documentation [113, 153][23] and examination of captured traffic using a program called SLparse that was created for the purpose. The SLparse program parses Second Life packets and tracks the state of circuits, this further described in section 3.2. A circuit can contain both reliable and unreliable packets. The reliable packets are acknowledged. If an acknowledgement is not received for a reliable packet it is re-sent.

TCP connections are also used by Second Life these are short lived and only carry a small amount of data (from traffic capture).

Typically between a client and a server there will be a single circuit. A client will speak to multiple servers each requiring a separate circuit for each one (from traffic capture).

The amount of traffic that occurs on these circuits varies considerably by type and depending on the activities that are being undertaken. Gaining a picture of how this works is part of the subject of this investigation.

4.3.3 Packets

Each packet has flags and a sequence number, then a variable length number which indicates the type of the packet [153]. The structure of the packet header is shown in figure 4.3. The first flag is called Zero and indicates if the body of the packet is run length encoded. The second flag is called Reliable and indicates if the packet needs to be acknowledged. The third flag is called Re-sent and indicates if the packet has been re-sent. The fourth flag is called ACK it indicates that the packet contains appended ACKs. The next four bits are unused. The next four bytes are the sequence number of the packet. The sixth byte indicates the number of extra bytes in the header. After the header is the number indicating the type of the packet. If the first byte of body is not equal to 255 then it is the packet type and it is a high frequency packet. If the first byte is 255 and the second is not then the first two bytes are the number and it is a medium frequency packet. If the first two bytes are 255 then the packet number is the first four bytes and the packet is of fixed or low frequency.

The packets in Second Life circuits have sequence numbers which are used to detect packet losses and track their quantity. The loss detection system is timeout based and operates at the receiving end of the connection. When a gap in the sequence space is detected and it remains unfilled for the appropriate amount of time a loss is assumed to have occurred. The timeout period is 16 times the average ping time. The system keeps track of the sequence number that it expects to see next. If this sequence number is seen then the value is incremented otherwise the sequence number is searched for in the list of sequence numbers of potentially lost packets. If the sequence number is found it is removed from the list. If it is not found and the packet is not marked as having been re-sent all of the sequence numbers between this one and the expected one are added to the list of possibly lost and the next expected sequence number is set to this packet sequence number plus one. This is known from examination of the source code of the client [112].

There are over 470 types of packet specified [24]. Each of these types corresponds to the function that the packet carries out. Example types of packet and their associated functionality include:

1. AgentUpdate: Which contains current camera information from the client to the server.

2. ObjectUpdate: Which sends the objects, including avatars and Prims

[23]Second Life: packet type templates `https://svn.secondlife.com/svn/linden/sandbox/2007/VWR-2972_1.18.2/scripts/messages/message_template.msg` [Online; accessed 10-November-2010]

[24]Second Life: packet type templates `https://svn.secondlife.com/svn/linden/sandbox/2007/VWR-2972_1.18.2/scripts/messages/message_template.msg` [Online; accessed 10-November-2010]

3. LayerData: contains information about the land, wind or clouds

4. Text message: Sends text for instant messaging

The round trip time is determined using special ping packets that the server and client respond to (from documentation and source). This information is used to determine the amount of time to wait for missing packets before they are considered lost at the receiving end. It is also used to determine length of time to wait for reliable packets to be acknowledged before resending them.

4.3.4 Adaptation to network conditions

MUVEs offer an immersive virtual environment. In order for this illusion to be preserved the system must react quickly and consistently to user input. The architecture of Second Life is such that in order for the user's avatar to react to input, messages have to be passed to and from the server [102]. When there is too much load on the server it changes the rate at which time passes on the island it represents, this information is sent from the server to the client. The number of avatars on the island effects both the load on the server and the amount of traffic that is sent to the clients controlling those avatars. A large number of avatars on an island can cause the interaction to become less responsive than the time dilation at the server would cause on its own.

There are rudimentary mechanisms for controlling traffic generation within Second Life. The client specifies the global rate at which it is prepared to receive data. This value defaults to 500 kbps and has a maximum value of 1500 kbps and a minimum value of 50 kbps. It may be set by the user, and recent clients [112] have some rudimentary facility for adjusting to packet loss (from the examination of the client source [112]).

There is a system in Second Life that limits the amount of bandwidth used. It will however usually remain at its default value, and will not probe the network for extra bandwidth. It can be adjusted by the user, who may choose values between 1500 Kbps and 50 Kbps for total bandwidth usage.

Throttle Adaptation

The Second Life client tracks the level of network congestion and adapts its bandwidth utilisation. Initially the cap on bandwidth may be set by the user to any value between 50 and 1500 Kbps. The initial value used is the smaller of 1500 Kbps and 1.5 times the user setting. During the session the bandwidth cap may range between 0.1 and 1.5 times the value set by the user. The value is calculated at the client and sent to the server, the server then uses this rate value when limiting the amount of bandwidth to use.

- When the loss rate is less than 0.5% and the current cap is less than the starting value the rate is increased by 0.1 times the set rate.

- When the loss rate is greater than 3% the sending rate is reduced by 0.1 times the set rate.

This is an additive increase and an additive decrease algorithm, which will not lead to convergence in transmission rates sharing the same network path, [164] by Ramakrishnan and Jain. The transmission rate is independent of round trip time. These properties of the congestion control system are not consistent with TCP and are not TCP fair [59].

4.3.5 Channels

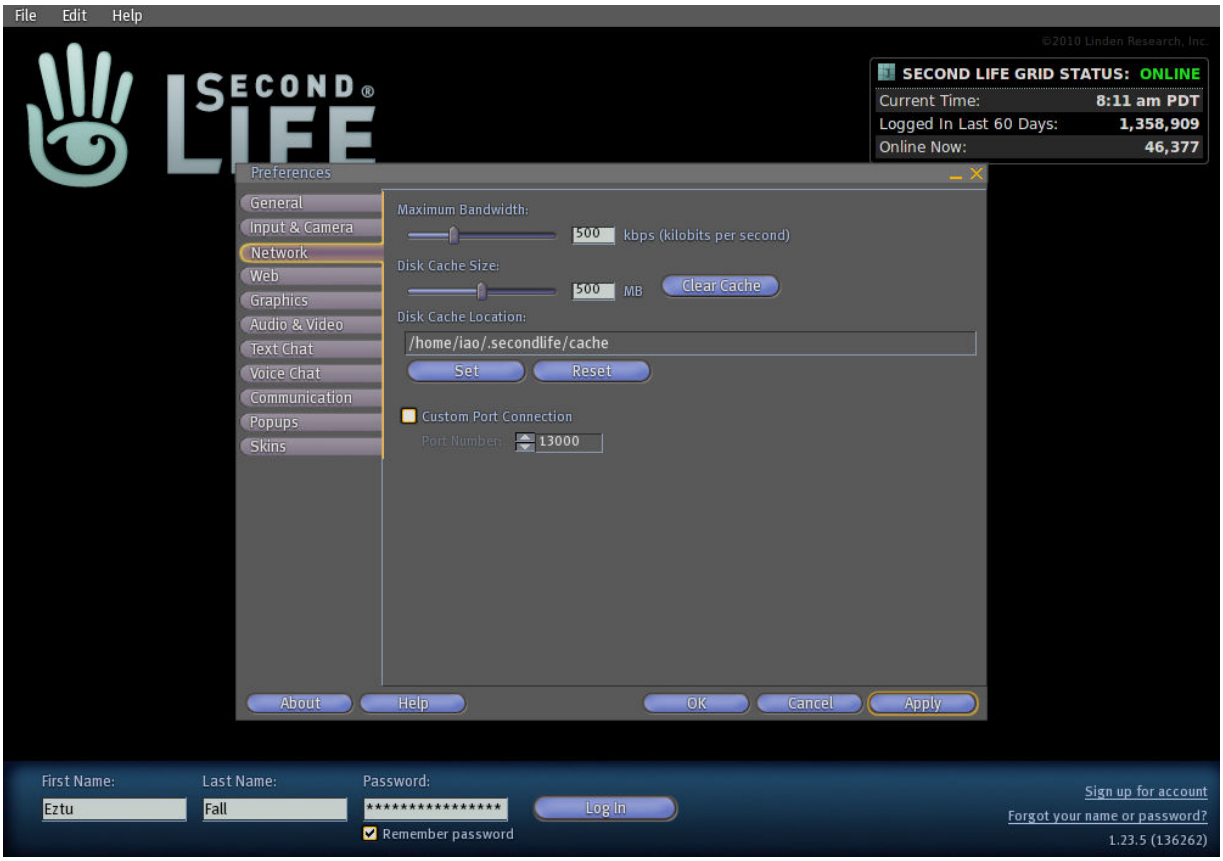
The traffic is split up into different channels which are given different amounts of bandwidth. There are seven sub categories of traffic each of which can be throttled individually [113].

1. Asset: Non physical information about the assets. An asset is a data resource identified by a unique ID.

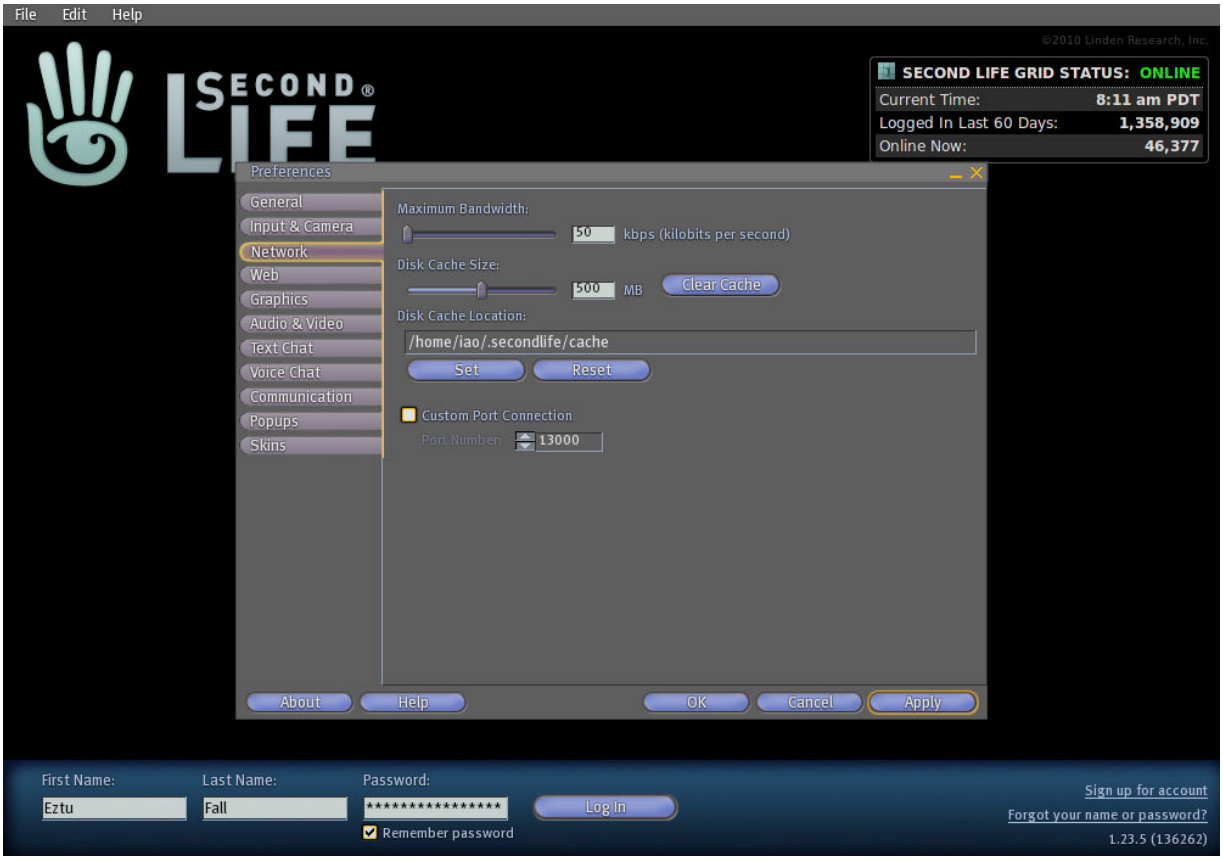
2. Task: Everything not covered by the other channels is included in this channel. This includes object update packets that include information about the movement of the avatars and the objects in the environment.

3. Wind: Wind LayerData packets. There is wind in Second Life that makes sound and affects other objects.

4. Land: Information about the terrain. The land and water that make up the contents of the Sim. Information about the land is stored as a height map. The land rarely changes on an island so this channel usually only contains packets when an avatar first sees an island or part of an island.

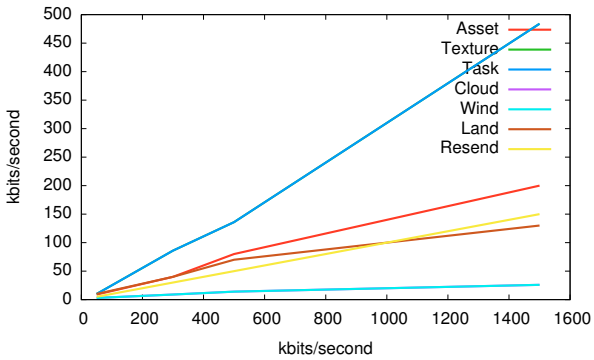


(a) 500 kbps

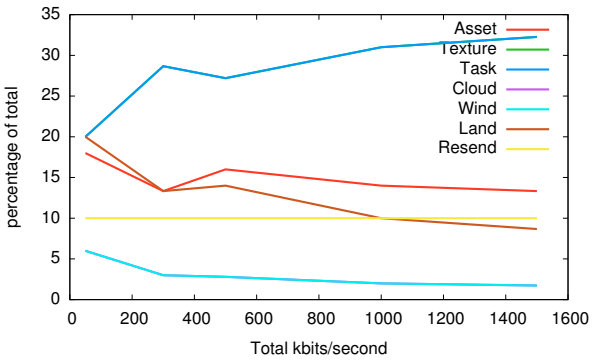


(b) 50 kbps

Figure 4.4: Second Life throttle settings.



(a) Bandwidth for each channel's throttle against the total.



(b) Percentage of the bandwidth for each throttle against throughput.

Figure 4.5: The amount bandwidth for each channel's throttle against the total.

5. Cloud: Cloud LayerData packets. The Sims calculate the amount and distribution of clouds in the Sim and communicate with the other Sims to maintain the relationship between the weather in the different Sims.

6. Texture: The images applied to objects are referred to as textures, when these are sent the Texture channel is used. In an environment with a large number of different objects there will be a large number of packets from this channel.

7. Resend: Some of the packets sent over a grid are marked as reliable, if one of these packets is not delivered it will be re-sent. When a packet is re-sent it not in the same channel as it would usually be, instead it is in the Resend channel.

The channels divide traffic into different categories depending on the element of the environment that the packets refer to. This makes sense from the perspective of categorising packets and traffic. There is a different importance to being able to interact with other avatars and to be able to see the environment in which the avatars are in. This system allows different parts of the environment to be prioritised, so that when there is little bandwidth the ground at least will appear quickly. The channel throttle values are calculated at the client from the total throttle value that is calculates. The client allows the user to set the overall throttle value but not the individual channel throttles. The amount of bandwidth given to each channel's throttle against total amount of bandwidth, the client is to use as the number of bits per second is show in figure 4.5a. The x-axis shows the total throttle value in Kbps and the y-axis gives the channel throttle bandwidth in Kbps. It is shown as a percentage total in figure 4.5b. The y-axis shows the percentage of the total bandwidth. The amount of bandwidth allocated to each channel's throttle differs least at the smallest value (50 kbps). The Task and Texture channels' throttles have the same value as the Land channel throttle at 50 kbps and the Asset channel throttle is only 1 kbps smaller, but at 1500 kbps Task and Texture are several times the value Land and Asset.

The Resend channel is given 10% of the total bandwidth, so when there is loss the reliable packets will make up more of the total bandwidth than when there is no loss.

This results in Texture, Land, Asset and Task packets getting mixed together. The different types of packet then cannot be given different amounts of bandwidth when it comes to resending them.

4.3.6 Avatar Control and Reaction

The user's avatar moves in response to input from the user that is sent to the server which then simulates the movement of the avatar and sends the information about the avatar including its position [102]. In order to allow the avatar to move smoothly the client program continues the movement and acceleration of the avatar without simulating any physics or any interaction from the user. The other objects and avatars are also updated in this manner. As the client continues the movement of the avatars with the information it has without any physics simulation the avatars can appear to be in places that it would be impossible for them to be in. The position that the client and server believe that the avatars and objects in the environment are in can therefore differ, with the server being authoritative on this information. The amount that these values differ will depend on how the avatars are moving and on the network

conditions. Flying consumes more bandwidth because the user is moving more quickly. However it rarely involves rapid change in direction so the position information will not contain large inconsistencies. Walking in a straight line will similarly not require rapid and consistent responses from the server. Walking inside a complex environment can involve lots of changes in direction which will require the faster response from the server.

4.3.7 Summary

Second Life is a popular MUVE on which this work focuses. Its is based on a client server model and has an open source client. The communication between client and servers in Second Life is accomplished with a custom application level protocol transported using UDP packets. The packets are arranged into circuits which contain sequence numbers which allow packet loss to be detected. The UDP protocol does not contain congestion control system. Second Life has a system to control packet transmission that changes in response to congestion.

The system in Second Life that limits the amount of bandwidth used usually remains at its default value, though it can be adjusted by the user. The system that adjusts the bandwidth utilisation based on network conditions will not effect the rate of packet transmission under normal circumstances. The traffic is split up into different classes which are given different amounts of bandwidth.

4.4 Comparison with Games Traffic

Multi player 3D games are similar to MUVEs in that they provide a 3D virtual environment. There are similarities in the types of information that a client and server exchange between games and MUVEs and differences. There are different types of multiplayer 3D games, that are different in the type of game, the view and type of control that the user has over their avatar or avatars.

There have been separate studies into network traffic of First Person Shooters [52, 74], Massively Multi player Online Role Playing Games (MMORPG) [24, 25], and Real Time Strategy (RTS) games [34, 185]. These are types of multi player games that are similar to MUVEs.

4.4.1 MMORPG

MMORPGS are a type of multi user game where users move around in a shared 3D virtual environment. There are methods of progress in these types of games. There are usually defined roles for the avatars that the user uses in the world, often come with goals for the user to achieve. The progress in game usually involves combat. This combat can involve different types of Non Player Character (NPC), ones that are computer controlled and have no function other than to fight the characters, ones that interact with the users in other ways and ones that are controlled by paid staff. The combat can also be between different players. Different user avatars usually have different skills and different amounts of ability. The type or role of the character often determines the skills that they posses though some games also allows the character to learn new skills.

Similar to MUVEs MMORPGs include social interaction and commerce. The money used in such games is usually acquired from battles, though may also have been purchased for real currency. MMORPGs often allow the user's avatar to be customized. The commerce in such games relates to the game, so items purchased will help the character in their progress through the game.

Chen et al. examine the network traffic of a game called *ShenZhou Online*[25], a MMORPG, in [24]. The packet size, bandwidth usage and inter arrival time were examined. The packets sent by this game are small 98% of the packets are less than 71 bytes. The bandwidth usage of this game is 7 Kbps. The inter arrival time of the packets very depending on events in the game. Kim et al. examines the network traffic of a game called *Lineage II* [26], a MMORPG, in [96].

Chen et al. examine the effect of network service quality on the enjoyment of players of online computer games, and so the length of time that they spend in the game in [25]. The network latency, jitter and loss rate effect the length of time a user spend online, but to different degrees. A formula was created to predict the length of time that players would spend online.

Svoboda et al. describes the analysis and modelling of the traffic of the Massively Multiplayer Online Game (MMOG), World of Warcraft (WoW), in [186]. The traffic they analysed was gathered from two sources. The first by

[25]ShenZhou Online `http://www.ewsoft.com.tw/` [Online; accessed 03-October-2010]
[26]NCsoft. Lineage II `http://www.lineage2.com/`, 1 October 2003. [Online; accessed 03-October-2010]

monitoring the at the players machine, the second is from the METAWIN project. This allowed the session times to be measured. It was discovered that the PSH flag is set on most packets captured, as the data units being sent are, mostly, smaller than the Maximum segment size (MSS) of the network and timing is important. When there is a large amount of interaction in the game environment the used bandwidth can increase to several times the normally used amount. A Network Simulator 2 (NS2) [27] script was created modelling the behaviour of WoW.

The results of a questionnaire to gain insight into what users think of the Internet in relation to games are presented by Oliveira and Henderson in [140]. The paper looks at MMORPGs as they provide the user with an immersive third person prospective in a multi-user environment. The questionnaire asked about the importance of computer games to the respondent, the answer showed that these users played more computer games than the general population. The second set of questions asked about the respondents' insertion in the game and indicated that most became immersed in the reality of the game, this illusion however was affected by the disruption from the real world. The majority of the respondents would prefer to be informed of changes in the network delay whilst in game. Less than half of respondents said that they would be willing to pay more for Quality of Service (QoS).

Henderson looked at the effect of network on the in game behaviour players of online network games in [74].

There have been studies on latency and user satisfaction. Henderson [75, 140] shows Half-Life players choose servers with low latency. Chen et al. in [25] shows that network latency, jitter and loss rate all effect the length of time a user spends online. Ruddle et al. looked at the latency of World Wide Web (WWW) applications is in [170].

Henderson shows that latency is important to the user experience in [74] and it is shown that the accuracy with which tasks can be completed within 3D environments in [35]. Environments where the avatar is controlled from a first person perspective (as metaverses are) are more sensitive than environments where the user has a global perspective. The tasks being undertaken also effect sensitivity to latency. Simple navigation through an open terrain is less sensitive than navigation through a complex environment [35]. The wide range of activities that metaverses aspire to support means latency sensitive tasks are encompassed.

The traffic of MMORPGs has the following properties. Most of the packets are small, for ShenZhou 98% of the packets are less than 71 bytes. The average bandwidth is about 7 kbps [186, 24]. The majority of MMORPGs use TCP for transporting there data. This traffic consists mostly of information about the avatars in the environment. The bandwidth is low but the number of packets is higher than this amount of bandwidth would suggest putting more stress on the network. The users have some sensitivity to network conditions.

4.4.2 Third Person Omnipresent

Third Person Omnipresent (TPO) are games where the user control a number of avatars. Real Time Strategy (RTS) games are strategy games where there are no turns, they are TPO. The user controls one or more characters in a virtual environment in combat with other characters. There are both multi player and non multi-player RTS games. The player is usually given isometric perspective or a free-roaming camera in newer 3D games.

Sheldon et al. examine the effect of latency on user performance in Warcraft III in [174]. User activity was separated into the components of explore, build and combat. This game does not appear to be sensitive to changes in the RTT. Claypool examines the effect of latency on user performance in Real-Time Strategy games in [34].

Nichols and Claypool [136] the effect of latency on an American football game: "Online Madden NFL Football", shows that there is a negative correlation between performance in the game and latency. This paper examines how running and passing are affected by latency, with higher latency producing worse results.

The traffic of TPOs has the following properties. The majority of the packets are small, for Online Madden NFL Football the median size of the packets is 77 bytes and for Warcraft III packets are around 46 bytes. The importance of latency is less than the other types of games.

4.4.3 First Person Shooters

In a First Person Shooter (FPS) the user controls an avatar and sees the virtual environment from the perspective of the avatar. These games usually involve shooting at the other avatars. The worlds used in these games are usually small and only allow a limited number of users. Not all FPSs are multiplayer games. When the game is multiplayer the type of activities that the game may involve. The simplest type of competition this type of game can involve is

[27] S. McCanne, S. Floyd, and K. Fall. ns2 (network simulator 2). `http://www.isi.edu/nsnam/ns/index.html`, 2005. [Online; accessed 15-October-2010]

the deathmatch where the players shoot at each other and the player with the largest number of kills is the winner. The games can be more cooperative involving teams of players shooting at each other.

The results of a questionnaire to gain insight into what users think of the Internet in relation to games are presented by Oliveira and Henderson in [140]. The paper looks at FPSs as they provide the user with a first person prospective in a multi-user environment, these are not the same as MUVEs as they have fixed environments and tighter time constraints. The questionnaire asked about the importance of computer games to the respondents, the answer showed that these users played more computer games than the general population.

The effect of variation in quality of service on the behaviour of players of network games is examined by Henderson and Bhatti in [75]. The paper examined the FPS, Half-Life[28]. It was discovered that when player comes to pick a server to join they will pick a server with a lower network delay. This paper does not show a significant correlation between the increase in delay during the game and the length of time that a player stays in the virtual world.

Feng et al. examine the network traffic of a game called *Counter-Strike* [29] in [51]. The packet size is small less than 60 bytes for incoming and between 0 and 300 bytes for outgoing.

The network behaviour of, the FPS, Quake3 [30] is examined and modelled by Land et al. in [106]. The server sends out packets at a mostly constant rate with the packet length depending on the map and the number of players in the game. The length of packets sent by the client does not depend on any of the observed parameters, the packet transmission rate however depends on the map and the graphics card of the client machine. An NS2 [31] model was created ignoring the effect of the map being changed and assuming a good graphics card for the game.

Claypool and Claypool give a comprehensive survey of the effect of latency on 3D games in [35]. They find first person avatar games require shorter RTTs than Third person avatar games and that Third person avatar games require shorter RTTs than Omnipresent games.

The traffic of FPSs is usually carried by UDP packets and has tighter than other types of game. They have small packets and low bandwidth.

4.4.4 Summary

As 3D games and Second Life are both 3D environments it is not unreasonable to assume that QoS requirements of metaverses are similar to 3D games. The affinity is closer with games that involve first person avatar control. In fact, one of the popular activities undertaken in Second Life is playing 3D games, examples include; New Babbage "a steampunk city ... a place they can role play and be creative"[32], "I Am Legend: Survival " a multi player first-person shooter[33] and "The Amber Raceway", which supports track racing [34].

MUVEs have larger bandwidth requirements than games. The virtual world in a computer game is usually set, so is known to the client so it is not sent to the clients during the session. The traffic therefore contains information about the movements of the players avatars.

4.5 MUVE

Kinicki and Claypool give an authoritative comparison of Second Life and 3D game traffic in [97]. This considers packet size, inter-arrival time and throughput. Based upon classification in [35], one example from three genres is chosen. First Person Avatar (Unreal Tournament 2003 [11][35]) 67 Kbps, Third Person Avatar (Madden NFL [136][36]) 14 Kbps and third person omnipresent (World of Warcraft [186][37]) 7 Kbps, compared to Second Life 775 Kbps. The

[28]Valve Corporation. Half-life. Sierra Studios, Electronic Arts and `http://store.steampowered.com/app/70/`, 19 November 1998. [Online; accessed 07-October-2010]

[29]Valve Software. Counter-Strike: A Counter-Terrorism Half-Life Modification. `http://www.counter-strike.net/`, 5 July 1995. [Online; accessed 03-October-2010].

[30]id Software. Quake 3 arena. `http://www.quake3arena.com/`, 2 December 1999. [Online; accessed 29-September-2010]

[31]S. McCanne, S. Floyd, and K. Fall. NS2 (network simulator 2). `http://www.isi.edu/nsnam/ns/index.html`, 2005. [Online; accessed 15-October-2010]

[32]`http://newbabbage.ning.com`

[33]`http://iamlegendsurvival.warnerbros.com`

[34]`http://www.amberracing.net`

[35]Epic Games and Digital Extremes. Unreal Tournament 2003. `http://www.unrealtournament2003.com/`, 2003. [Online; accessed 29-September-2010]

[36]EA Tiburon. Madden NFL 2004. EA Sports `http://www.easports.com/games/madden2004/home.jsp`, 2004. [Online; accessed 03-October-2010]

[37]Blizzard Entertainment. World of Warcraft [Computer game]. `http://us.battle.net/wow/en/`, 23 November 2004. [Online; accessed 04-April-2011]

bandwidth requirements of Second Life are around ten times that of 3D computer games, in addition Second Life has larger packets. The authors conclude; *"this large turbulence suggests that meeting the quality of service requirements of Second Life, which are likely to be similar to that of third person avatar games, is a challenge."* [97].

There has been other work to investigate the behaviour of Second Life traffic. Fernandes et al. in [53] look at the Second Life traffic from the network and connection level. It looks at different scenarios looking at the amount of data and packets, also the packet size, transport layer type, and direction. Kinicki and Claypool in [97] look at the effect of different actions in Second Life on its traffic looking at the same actions as [53] and also looking at teleportation. It looks at packet size, inter-arrival time and bandwidth usage and compares these to the figures for online games. Antonella et al. look at Second Life traffic with respect to the construction of a synthetic traffic model for Second Life in [6]. It looks at different scenarios and properties of the traffic. It also looks at the application level packet types to determine which are more popular and what their purpose is.

Second Life is looked at by Varvello et al. in [188]. The paper describes a Second Life client program that was created to investigate Second Life. It visited all of the publicly accessible island to determine the number, the number of avatars and objects in the island. It looks at the amount in the object on an island and discovered that the objects on most of the islands change very little or not at all. Only 2% of the islands had more than 20 avatars and 45% had no avatars, with more than 90 being very rare. Most avatars stand in groups and act similarly to real people that gather in specific areas. It suggests that the use of a client side cache for the objects, that prioritises objects that are unlikely to be deleted in the near future. The use of prefetching is also suggested, where information about areas which already have a large number of avatars would be sent to the client. The use of a peer-to-peer architecture is advocated where the clients share a cache and information about the avatars.

Second Life contains a system to limit the amount of data that it transmits. This value is determined at the client by the user. The bandwidth allocated by the client is divided up among a number of different streams. The system works by limiting the amount of data that can be sent per unit time. The system keeps a count of this for each unit time and counts the amount of data that has been sent out for all the streams and for each stream individually. After the data has been queued, to be sent, the counters of the amount sent are reset. The values for the bandwidth used are examined, and then the values are adjusted so that spare capacity is given to the streams that need more capacity by removing capacity from streams that are not using it.

Second Life requires traffic to send and receive updates on the position of the avatars and any attached Prims, this requirement will be similar to, though possible larger than, the requirement of a MMORPG for the same information. The difference is that Second Life also has other traffic that relates to the environment. To get similar utility from the network the same type of traffic from the two types of systems require similar amounts of network resources. The information about the avatars fits into Task channel along with other traffic relating to the environment (from documentation and OpenSim [131] source [132]). A game with a fixed environment may also have traffic similar to the traffic that is contained in the Asset channel and small amount of Texture data. Different users performing different tasks in the virtual environment will have different priorities for the avatars and the non avatars, it would be useful if a system was included to adjust the priority between these two things.

The amount of bandwidth necessary for the systems to be usable depends on the environment that the avatar is in. The method of reducing the amount of bandwidth used by the system affects its behaviour. The user sets the amount of bandwidth desired and the system sets the individual channel's throttle from this value. The percentage of the total bandwidth that is given to each channel depends on the amount of available bandwidth (from the client source [112]).

The current channels separate out the different types of data that the client and server communicate between each other. They however do not separate data that would have different level of importance to the user with respect to usability and responsiveness of the system. The movement of the user's avatar, the movement of the other avatars and the other objects all have different levels of importance to the user. The user's avatar not responding to input promptly will reduce the interactivity and immersiveness of the system. The importance to the user of being able to see the other avatars moving around is also likely to be higher than the importance of being able to see the contents of the environment. The bandwidth necessary for giving up to date information about the avatars in the environment will vary only with the number of avatars. A channel system should take into account that this requirement exists and give avatar traffic the amount of bandwidth that it requires. The Wind and Cloud channels have only small amount of traffic that doesn't vary. The Land channel is only used when a new island is encountered, this traffic requires a high priority because if this traffic doesn't arrive there won't even be land. The Texture channel uses a lot of the bandwidth when textures are being sent to the client, this is important for the appearance of the environment but not often for usability. The Task channel contains the information about the avatars that is important for the interactivity and also

the information about all of the Objects that is important so that the user can see the Objects in the environment. The Resend channel contains all of the re-sent packets.

This leads us to ask whether it would be better with a system that adjusts to the amount of available bandwidth rather than relying on the user. There is also the question of whether the channels that the traffic is currently divided into are appropriate for Second Life or MUVEs in general. This is further discussed in chapter 8. An improved channel system would therefore have:

1. Asset: Containing the contents of the Asset channel.
2. Avatar: Containing all of the avatar traffic.
3. Task: Containing the remainder of the Task channel.
4. Layer: Containing the Wind, Land and Cloud channels.
5. Texture: Containing the contents of the Texture channel.

With the re-sent packets belonging to the same channel as the original sending of the packets.

4.6 Conclusion

The number of high bandwidth applications being used on the Internet has been increasing recently. Much of the existing traffic as well as much of projected new traffic uses Transmission Control Protocol (TCP) as its transport level protocol [61, 89] and as a result it is congestion controlled. However, a significant proportion uses User Datagram Protocol (UDP), which has no provision for congestion control, consequently congestion control for non TCP traffic has been and will continue to be an issue of some importance for the Internet.

The focus of this chapter is on Multi-User Virtual Environments (MUVEs) a relatively new class of applications. They provide computer based simulated environments where multiple users interact using avatars. The rules of the virtual environment are usually similar to the real world, in that it is not possible to walk through walls and things usually fall downward. MUVE usually only refers to non-gaming environments or environments that are not exclusively for gaming [129]. Most MUVEs use UDP as their primary transport layer protocol. They mostly implement their own rudimentary congestion control systems.

A TCP fair adaptive traffic management scheme will detect network conditions and control the number of packets transmitted so that it competes fairly with other connections on the network and prevents congestion. Congestion control is important because it prevents congestion collapse [81] and allows network resources to be used fairly [86] and efficiently [164].

As MUVEs simulate an environment that is similar to the real world, that users can become immersed in, it is important that the system reacts to users input quickly and consistently, MUVEs have soft real time constraints. A MUVE with an adaptive traffic management system will be able to change the amount of traffic that it sends out onto the network depending on the amount of other traffic and congestion on the network. In a congested network the round trip time will increase, which will adversely affect the experience of using a MUVE. This thesis will show that a traffic management system created specifically for MUVEs will be better able to react to its needs, including being able to prioritise things other than maximum throughput.

In a traditional 3D multiplayer environment, such as a game the environment is unchanging and known to both the server and the client, usually before the session begins [34, 11]. This allows the network traffic to consist of the actions of the players of the game and any non-player characters in the environment [110]. The server and clients can then share a model of the environment. Many Massively Multiplayer Online Role-Playing Games (MMORPGs) have long sessions where users join and leave during the session. These games however usually do not allow the users to make any permanent changes to the environment. First person shooters (FPS) [106][38][39][40][41][42][43] often allow the users

[38] Bytonic Software. Jake 2. `http://bytonic.de/html/jake2.html`, 05 May 2006. [Online; accessed 22-October-2010]

[39] Epic Games and Digital Extremes. Unreal Tournament 2004. Atari `http://www.unrealtournament2003.com/ut2004/`, 16 March 2004. [Online; accessed 28-March-2011]

[40] id Software. Quake 2. `http://www.idsoftware.com/games/quake/quake2/`, 9 December 1997. [Online; accessed 29-September-2010]

[41] id Software. Quake 3 arena. `http://www.quake3arena.com/`, 2 December 1999. [Online; accessed 29-September-2010]

[42] idSoftware. Quake. GT Interactive `http://www.idsoftware.com/games/quake/quake/`, 22 June 1996. [Online; accessed 22-October-2010]

[43] RavenSoftware. Quake 4, 2005. `http://www.idsoftware.com/games/quake/quake4/` [Online; accessed 30-June-2011]

to create new maps in which to play, this is done using external tools [125][44][45] and the map must be created and distributed to the players before the session begins. In the non-gaming virtual environments like Second Life (SL) on the other hand the world is not known to the client before the user joins the session as the world can be changed by the users from inside the environment.

As the contents of the world are not known to the client before it connects to the server running the world, it must receive all of the information about the world whilst connected. This can result in the bandwidth requirement for such applications being an order of magnitude larger than those of a FPS or MMORPG [53, 6, 97].

It also makes the traffic more varied. In addition to the avatars; MUVEs also have other objects and terrain that can change. This information has to be sent to the client, however the relative importance of these things is not proportional to the amount of traffic that they require. This suggests that there is a case for treating different sorts of traffic differently. For example prioritizing traffic that is related to avatar control over that associated with drawing back ground traffic.

Such a world is not usually limited in size or number of simultaneous users; to support this, the world is usually split up into a number of different servers. So long as the users do not gather in a small number of places, dividing the world into sections will work. This is the technique used by Second Life [113]. This means that as the user moves between areas in the game the client has to communicate with more than one of the servers.

There are a number of different MUVEs that were created for different purposes [131, 115, 117, 92, 62, 3]. They differ in the kind of environments that they provide and the number and type of users that they support. The data model used also differs between the different systems [176, 53]. Systems where the world lasts for a long time but users will join and leave usually use an architecture structure where there is a server that performs the simulation.

In summary: the requirements of a Multi User Virtual Environment differ from those of traditional data applications in that they do not require every packet to be delivered. They are resilient against low levels of packet loss. At the same time they have similarities to applications like video conferencing in that some of their traffic is inelastic and is sensitive to delay, they also have traffic that does not have these properties. Consequently, for much of the data transmitted by a MUVE TCP is not an appropriate transport protocol. In this MUVEs are similar to multi-player games, whether they be first person shooters like Quake 2 [46] or massively multi-player online games like World of Warcraft[47]. However, there is an important difference, the bandwidth requirements of MUVEs can be one or two orders of magnitude greater than other classes of games. Therefore it is important for the stability of the network that they contain effective traffic management and congestion control facilities.

[44] Raphal Quinet and Brendon Wyber. Doom editing utilities. `http://www.montefiore.ulg.ac.be/~quinet/games/DEU/DEU-en.html`, 26 January 1994. [Online; accessed 08-October-2010]

[45] Epic Games and Digital Extremes. UnrealEd. `http://udn.epicgames.com/`, 2004. [Online; accessed 22-October-2010]

[46] id Software. Quake 2. `http://www.idsoftware.com/games/quake/quake2/`, 9 December 1997. [Online; accessed 29-September-2010]

[47] Blizzard Entertainment. World of Warcraft [Computer game]. `http://us.battle.net/wow/en/`, 23 November 2004. [Online; accessed 04-April-2011]

Chapter 5

Second Life Evaluation

In order to gain insight into the nature of Second Life (SL) traffic and how it relates to the tasks or activities being undertaken by avatars in-world, measurements were taken and experiments carried out. Firstly, the global level of traffic is for numerous common activities and how these might relate to throttling of traffic at a global level were established. Secondly, how traffic is distributed between circuits, channel throttles and packet types were established. Thirdly, changes in the traffic mix with different environments, events and activities undertaken in-world were correlated.

The traffic during a session of users using Second Life was captured giving the traffic characteristics under real world conditions. An experimental investigation was then undertaken to further investigate the traffic characteristics of SL.

Section 5.1: The related work is discussed.

Section 5.2: An analysis of Second Life traffic captured during a user session is presented.

Section 5.3: An experimental investigation of Second Life is presented.

Section 5.4: Based on the results the network requirements of Second Life are presented.

5.1 Related Work

There have been several previous studies into SL Traffic. Two papers analyse client side traces of traffic generated for islands with different concentrations of objects and avatars, for defined avatar activities [53, 97]. Other studies have used avatar crawlers [188, 103] to collect server side statistics on loads and usage patterns. The subject of creating models of Second Life traffic is treated in [6].

5.1.1 Measurement of SL Traffic

Fernandes et al present measurements of Avatars Standing Walking and Flying in popular and unpopular places in [53], where popularity is measured by the proportion of time avatars spend on the island. Kinicki and Claypool add teleporting to the list of activities measured in [97]. They also separately quantify the effect of the density of objects and number of avatars on throughput. Whilst some throughput measurements are larger than [53] they are in the same order of magnitude. Walking in a crowded, dense place requires and average bandwidth of 775 Kbps and Teleporting requires 1164 Kbps. Both the island and avatar action have a marked effect on the bandwidth. The authors note that the magnitude of the effect of the island is 2.5 times the amount of bandwidth in the popular island compared to the unpopular in the first study and 7 to 9 times in the second.

Varvello et al. in [188] present comprehensive server side statistics which show that total network load follows a diurnal pattern ranging from a low of 1.7 million packets per second to a peak of 3.2 million packets per second. An average packet size of 500 bytes is assumed based on [53]. This gives a total load of 18.9 Gbps and an average of 280 Kbps per client. 50% of user sessions lasted ten minutes or less. Avatars spent 80% of their time standing with the rest divided between Walking, Running, Flying and Teleporting. Walking and Teleporting accounting for more than 90% of the remainder.

La and Machiardi show that avatar movement in Second Life [103] mirrors that of people in the real world. An analysis of textures in Second Life is presented in [102]. Individual and aggregate texture sizes are quantified, with a median size of 95 Kb and the studied regions having between one and four thousand textures.

Liang et al. present an analysis of Second Life network traffic in [109]. This is based upon two three minute sessions connecting to the Swedish Embassy. They observe that during the second run the SL cache significantly reduced the bandwidth.

The construction of a synthetic model for SL traffic is advocated by Antonello et al. in [6]. It is suggested that the ability to model SL traffic would help game developers and network providers. Models of inter-packet arrival time and packet size distributions were derived for standing and walking in popular and unpopular places.

5.1.2 Comparison with Games

The literature regrading multi player games is discussed in section 4.4. There have been studies into network traffic of First Person Shooters [52, 74], Massively Multi player Online Role Playing Games (MMORPG) [24, 25], and Real Time Strategy (RTS) games [34, 185].

Kinicki and Claypool given a comparison of Second Life and 3D game traffic in [97]. They consider packet size, inter-arrival time and throughput. Based upon classification in [35], one example from three genres is chosen. First Person Avatar (Unreal Tournament 2003 [11][1]) 67 Kbps, Third Person Avatar (Madden NFL [136][2]) 14 Kbps and third person omnipresent (World of Warcraft [186][3]) 7 Kbps, compared to Second Life 775 Kbps. The bandwidth requirements of Second Life are around ten times that of 3D computer games, in addition Second Life has larger packets.

As 3D games and Second Life are both 3D environments it is not unreasonable to assume that QoS requirements of metaverses are similar to 3D games. The affinity is closer with games that involve first person avatar control. In fact, one of the popular activities undertaken in Second Life is playing 3D games, examples include; New Babbage "a steampunk city ... a place they can role play and be creative"[4], "I Am Legend: Survival" a multi player first-person shooter[5] and "The Amber Raceway", which supports track racing [6].

There have been studies on latency and user satisfaction. Nichols and Claypool [136] the effect of latency on an American football game: "Online Madden NFL Football", shows that there is a negative correlation between performance in the game and latency. Henderson [75, 140] shows Half-Life players choose servers with low latency. Chen et al. in [25] shows that network latency, jitter and loss rate all affect the length of time a user spends online. Ruddle et al. looked at the latency of World Wide Web (WWW) applications in [170].

Claypool and Claypool give a comprehensive survey of the effect of latency on 3D games in [35]. Henderson shows that latency is important to the user experience in [74] and it is shown that the accuracy with which tasks can be completed within 3D environments in [35]. Environments where the avatar is controlled from a first person perspective (as metaverses are) are more sensitive than environments where the user has a global perspective. The tasks being undertaken also effects sensitivity to latency. Simple navigation through an open terrain is less sensitive than navigation through a complex environment [35]. The wide range of activities that metaverses aspire to support means latency sensitive tasks are encompassed.

[1] Epic Games and Digital Extremes. Unreal Tournament 2003. `http://www.unrealtournament2003.com/`, 2003. [Online; accessed 29-September-2010]

[2] EA Tiburon. Madden NFL 2004. EA Sports `http://www.easports.com/games/madden2004/home.jsp`, 2004. [Online; accessed 03-October-2010]

[3] Blizzard Entertainment. World of Warcraft [Computer game]. `http://us.battle.net/wow/en/`, 23 November 2004. [Online; accessed 04-April-2011]

[4] `http://newbabbage.ning.com`

[5] `http://iamlegendsurvival.warnerbros.com`

[6] `http://www.amberracing.net`

Game	**Type of Game**	**Average bandwidth**
Unreal Tournament 2003	First Person Avatar	67 Kbps
Madden NFL	Third Person Avatar	14 Kbps
World of Warcraft	Third Person Omnipresent	7 Kbps

Table 5.1: The average bandwidth usage of different type of game.

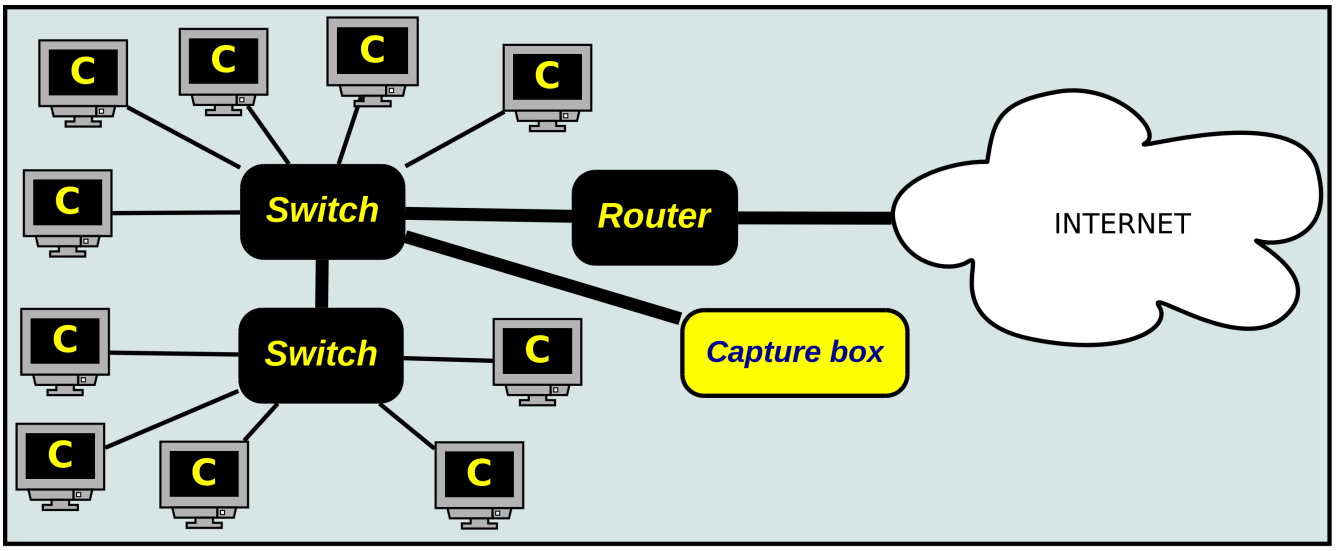


Figure 5.1: The structure of the lab network.

5.2 Observing SL in the wild

This study contributes to our understanding of SL traffic as previous work has focused on measuring specific scenarios rather than actual usage. It also facilitates comparison between the demands placed upon the network by SL with Transmission Control Protocol (TCP) Fair behaviour.

During a session, on the 13/11/2008, to introduce 30 people to Second Life the traffic going to Linden labs was captured to allow it to be examined. The workshop was for 30 faculty staff on using metaverses for teaching, most of the users had not used Second Life before. The traffic was captured by mirroring the uplink port on the switch for the lab to machine capturing the traffic. The capture was started a little after 9 and the lab started at 9:30. The participants went for lunch at 12:20. The hands-on session was from 10:10 until 10:45. After the hands-on session there was an opportunity for the participants to use Second Life. The timetable for the event is shown in table 5.2.

The users were given user names and passwords, for Second Life, to use during the lab session. During the hands-on session the users were first asked to login using either the password they were provided with or a new password that they created. They were then told about the different methods of communicating in Second Life. The next lesson was on moving around in Second Life, they then learnt how to teleport between different islands. The example given to teleport to was Minerva Island.

The workshop showcased some existing projects prior to a hands-on session during which all thirty participants were concurrently active within SL. The activities focused on exploration and communication. After the hands-on session, there was a break followed by some presentations.

9:00	Coffee and introductions	
9:30	Welcome	
9:35	Virtual Worlds – the Potential for Education	
10:10	Using Second Life (**The hands-on session**)	This was when all the users were on at one time.
10:45	Break: Tea, Coffee and Biscuits	
11:00	Minerva Island: Second Life at St Andrews	
11:45	Creating content for Second Life	
12:15	Panel Session with Questions and Answers	
12:30	Lunch and Project Demos	

Table 5.2: The timetable for the lab session

5.2.1 Methodology

The lab computers are connected to 2 100 Mbit switches which have 1 Gbit connections to a router and from there to JANET [177]. The traffic traces were captured using passive monitoring [122] techniques. The traffic was captured by mirroring the up-link port on the switch to the machine capturing the traffic. The structure of the network is shown in figure 5.1. The capture was started a little after 09:00 and the introductory session started at 09:30. The participants went for lunch at 12:20. The hands-on part of the session was from 10:10 until 10:45. The capture was finished at 13:45.

Packets were captured using tcpdump and then analysed using SLparse which uses libpcap to read the trace file, this program is described in section 3.2 in chapter 3. There is a trace route to one of the machines at Linden Labs in Appendix D.

The SLparse program was developed to analyse Second Life circuits. SLparse detects different connections by their source and destination addresses. It generates graphs for information extracted from the trace. TCP connections are also detected and their packets grouped to allow the graphs to include information about TCP connections.

When a packet is received its Internet Protocol (IP) header is examined to determine the transport level protocol used by the packet. The port numbers and IP address are used to distinguish the connections. An existing connection's data structure is then searched for. If one does not exist then it is created and if a connection does exist then it is updated. For TCP connections only a small amount of information about the state of the connection is recorded.

SLparse tracks the state of Second Life circuits. It groups together packets that belong to the same circuit and treats them like connections for the purposes of the functions not specific to Second Life traffic. Support is included to parse the headers of packets and print this information. The packet type information is then used to group packets into the channels and generate graphs from this information. Information about the size of packets for different types of packet and for different circuits can be extracted by the program and the per-packet type information can be grouped per packet.

For each connection a data structure is created to hold statistics about the connection. The statistics common to all types of connection are calculated separately from the statistics specific to the different types of connection. The number of bytes and packets a second is calculated at different granularities for the different connections. The distribution of packets sizes is calculated for each connection and the fairness between different connections is calculated.

The number of packets and the number of bytes is counted for different intervals. The intervals used are 10 milliseconds, 100 milliseconds, 1 second and 1 minute. The number of packets and bytes a second is calculated from this. The numbers of bytes and seconds for the different protocols are also counted and output for the different time intervals. The supported protocols are TCP, User Datagram Protocol (UDP), Datagram Congestion Control Protocol (DCCP) and Second Life circuits. The number of application levels bytes is calculated for the different units of time overall and per protocol. The maximum number bytes a second and the maximum number of application bytes a second is also calculated. The distribution of packet sizes is recorded and output as the Probability Distribution Frequency (PDF) and the Cumulative Distribution Frequency (CDF). The frequency distribution the number of bytes a second is recorded and output as the PDF. The time series graphs in this section show the time of the graph for which there are packets in the trace.

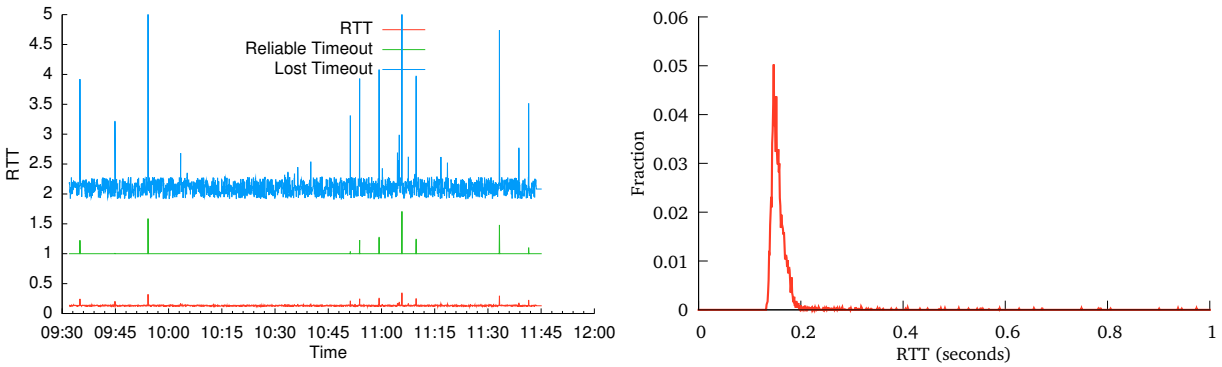
SLparse outputs a textual representation of the trace, that is similar to tcpdump, for the simple information about packets. The information about the state of the connection is also included in the output. The output of SLparse has been compared with that of other programs where they can both parse the same code and the code of SLparse has been checked through static analysis.

The traffic for different IP addresses in the lab, representing different computers, were separated out into different traces and analysed separately.

5.2.2 Experimental aims

The aim of this measurement was to gain insight into the interaction of several Second Life clients on the same network and to get greater insight into the nature of Second Life traffic during the normal use of Second Life. The amount of bandwidth that each client uses and the extent to which this bandwidth usage is limited by the throttle system.

There is a concern that Second Life and other MUVEs will use far more than their fair share of the available bandwidth and cause problems such as congestion collapse. To discover if this is true it is necessary to measure the amount of bandwidth Second Life is using and calculate the amount of bandwidth that it will use under different network conditions. This then needs to be compared against the amount of bandwidth that other systems using the



(a) The round trip time against time.

(b) Distribution frequency of round trip times.

Figure 5.2: Round trip time values.

network will use under those circumstances.

5.2.3 Network Conditions

To detect loss SLparse uses the method used to detect loss by Second Life. Second Life's loss detection system relies on time outs, so will detect the loss a little after it has actually happened. The number of packets lost is 3841, which is 0.02% of the total Second Life packets, which is due to the fact that the connections were only using a small fraction of the available bandwidth. Loss is ordinarily an indication of congestion, so the loss in this trace would then be a sign of congestion. The overall loss would indicate that there was very little congestion on this network, however the losses are grouped together.

Second Life determines the round trip time using special ping packets that the server and client respond to. This information is used to determine the amount of time to wait for missing packets before they are considered lost at the receiving end. It is also used to determine the length of time to wait for reliable packets to be acknowledged before resending them. Ping packets are sent out at a continuous rate and replies are sent back in response. When a response is received the time since that last ping was sent is calculated. If the response's number is different from the expected ping number the difference between the reply number and the expected number is multiplied by the time between pings being sent, one second. This value is then added to the calculated time. Slparse calculates round trip times (RTTs) from the traces by observing the ping packets and their replies. The lost timeout time is 16 times the RTT and reliable timeout is 5 times the RTT.

The round trip time against time for a connection in the trace is shown in figure 5.2a. The x-axis is the time of day and the y-axis is the time in seconds. The red line is the RTT values, the green line is the retransmission timeout value, blue line is the loss detection timeout value. The distribution frequency of round trip time value sampled once per second is shown in figure 5.2b. The x-axis shows the RTT in seconds and the y-axis shows the fraction of the values that are of each value. The average round trip time is around 150 ms. Using the average packet size of 304 KB and a loss rate of 0.02% for these connections gives a fair sending rate of 1455 kbps for a connection under these conditions. This is around twice the value that the clients are asking the server to use.

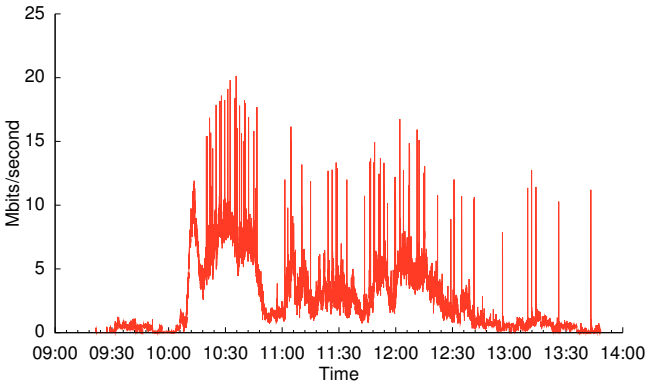
	Total	**Second Life**	**TCP**
Bytes (KB)	5,148,894	5,053,021	95,813
Packets	17,042,669	16,632,300	410,225
Kbps	2,569	2,521	47
Connections	22,192	332	21,853

(a) Summary statistics for the trace as a whole.

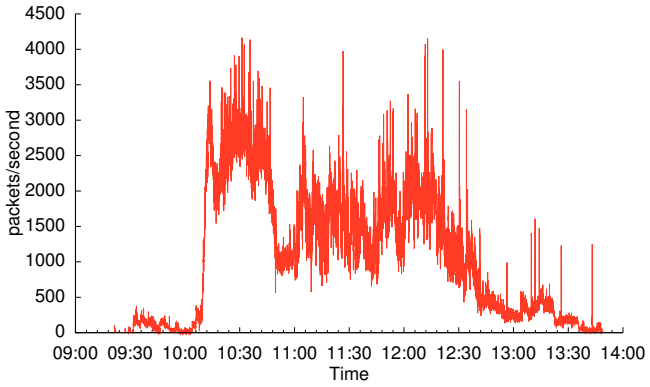
	Total	**Second Life**	**TCP**
Bytes (KB)	5,148,894	5,053,021	95,814
Packets	17,042,669	16,632,300	410,225
Kbps	2,569	2,521	47.8
Connections	22,192	332	21,853

(b) Summary statistics for the hands-on session.

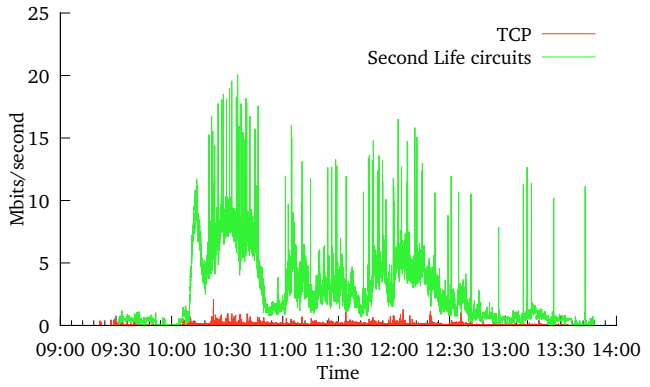
Table 5.3: Summary statistics for all the machines.



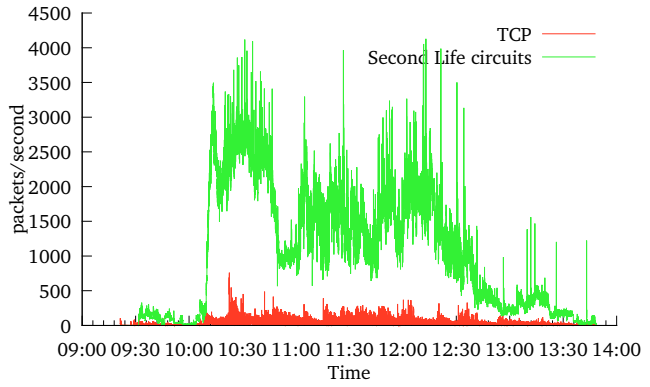
(a) The total number of bytes per second.



(b) The total number of packets per second.



(c) The number of bytes per second for each protocol.



(d) The number of packets per second for each protocol.

Figure 5.3: The aggregate packet and bytes per second.

5.2.4 Summary Statistics

First we present an overview of the traffic statistics before looking at particular aspects in more detail. The overall statistics for the session are in table 5.3a. During the workshop 5.1 Gigabits of data were transferred in over 17 million packets. Of that 98% was Second Life UDP traffic and 2% TCP traffic. The UDP traffic was distributed over 332 separate Second Life Circuits an average of around 11 for each client computer. The TCP traffic was distributed over 21,853 separate connections. The average amount of traffic per circuit was 15,219 KB. The average size of TCP connections were 4 KB. Thus the UDP traffic counts for the overwhelming majority of Second Life Traffic and it is organized into circuits that are both long lived and contain large amounts of traffic.

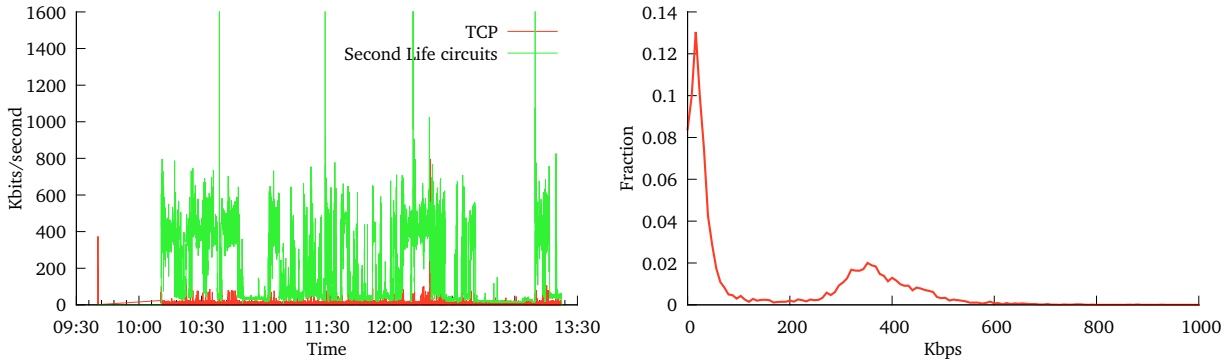
There were about 30 people in the lab, this works out to about 11 UDP circuits per person. The amount that people moved around and the locations that they visited will determine the number of connections that correspond to each person. The TCP connections are used for Hypertext Transfer Protocol Secure (HTTPS) [166], Second Life requires a small amount of secure communication. For each secure communication a new connection is created, this leads to large number of connections with only a small amount of data.

The summary statistics for three different lab computers are shown in table 5.4. These computers are referred to as machines 1, 2 and 3.

(a) The summary statistics for the connections involving machine 1 are shown in table 5.4a. The mean number of Second Life connections per machine is 10, this machine has more than double that, the mean number of TCP connections is 705 which is close to the number this machine has. The user of this machine likely moved around more than many of the other users.

(b) The summary statistics for the connections involving machine 2 are shown in table 5.4b. The different machines, so different users, use different amounts of bandwidth and form different numbers of connections.

(c) The summary statistics for the connections involving machine 3 are shown in table 5.4c. This machine has more



(a) Against time.

(b) Probability Distribution Frequency

Figure 5.4: The amount bandwidth used by Machine 3.

than double the mean number of Second Life connections, this machine has close to the mean number of TCP connections. The user of this machine likely moved around more than many of the other users.

The number of packets and bytes a second for the entire trace per protocol are shown in figure 5.3. In all of the graphs the x-axis is the time of day, in the left graphs y-axis is the bandwidth in Mbps and in the right hand graphs the y-axis is packets per second. At the beginning of the connection there are a number of TCP connections. These TCP connections are the clients downloading the welcome web page. The clients then log on to the system using HTTPS. The circuits are then created. The test network was a 100 Mbit Ethernet network connected to JANET, giving it far more bandwidth than was being used by this session. The largest amount of traffic is during the hands-on session when all of the users were using the system.

The throughput against time by machine 3 is shown in figure 5.4a. It can be observed that there is significant variance in the throughput. This reflects the different activities being undertaken. The periods where there is low throughput correspond to periods of lack of activity by the avatar in Second Life. Conversely the periods of high activity correspond to higher throughput. In addition it can be assumed that there is some perturbation and jitter introduced by the network. Taking the network perturbation into account it appears that the traffic is being capped by the Second Life throttle system. The peaks of activity are focused around 400 Kbps with very few samples being above 750 Kbps.

The PDF of bandwidth utilization of machine 3 is shown in figure 5.4b. It is a binomial distribution, in the absence of activity the throughput required appears around 20 Kbps. During active periods the peak value is just short of 400 Kbps. This suggests that the bandwidth utilization is being capped.

There are a number of features that are of interest:

- At the start of the graph there is little activity.

	Total	**Second Life**	**TCP**
Bytes (KB)	169,795	166,191	3,603
Packets	466,776	451,689	15,087
Kbps	168	165	3.6
Connections	823	24	799

(a) Summary statistics for machine 1.

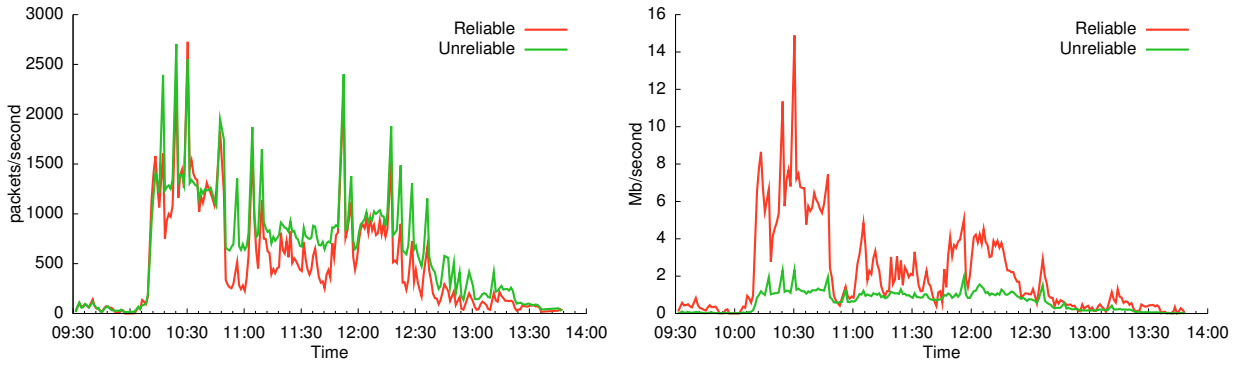
	Total	**Second Life**	**TCP**
Bytes (KB)	143,113	140,350	2,763
Packets	484,791	473,802	10,989
Kbps	90	88	1.7
Connections	589	9	580

(b) Summary statistics for machine 2.

	Total	**Second Life**	**TCP**
Bytes (KB)	268,590	264,605	3,985
Packets	785,810	767,997	17,813
Kbps	161	159	2.4
Connections	976	19	957

(c) Summary statistics for machine 3.

Table 5.4: Summary statistics for individual machines.



(a) Packets

(b) Bytes

Figure 5.5: The number of reliable and unreliable packets per second against time.

- At the beginning of the connection there are a number of TCP connections. These TCP connections are the clients downloading the welcome web page. The clients then logs on to the system using HTTPS.
- The circuits are then created.
- There is a prolonged peak of network usage from around 10:10 to 10:45. This corresponds to the hands-on session, where each of the participant was asked to complete a number of tasks within Second Life.
- After this there is prolonged significant activity up to 12:30 when the workshop ended.
- A few people stayed behind to experiment with Second Life until 2:00pm

The period of high activity between 10:10 and 10:45 represents a lab full of users simultaneously using Second Life and completing tasks within it. Later activity is less intense.

The summary statistics for the hands-on session part of the trace are shown in table 5.3b. During this period the average transmission rate per user is a lot higher than the average for the entire session.

The number of packets per second in the circuits that are reliable and the number that are unreliable are shown in figure 5.5a. The x-axis is the time of day and the y-axis is the number of packets per second. The number of reliable and unreliable bytes are shown in figure 5.5b. The x-axis is the time of day and the y-axis is the bandwidth utilization in Mbps. The red line is the packets marked reliable and the green line is the packets not marked reliable. The reliable packets appear to be larger than the unreliable and more so when there are more packets. The reliable and unreliable bytes and packets both increase during the hands-on session.

5.2.5 Packets

The distribution of packet sizes are shown in figure 5.6. The x-axis is the size in bytes and the y-axis is the fraction of the total samples.

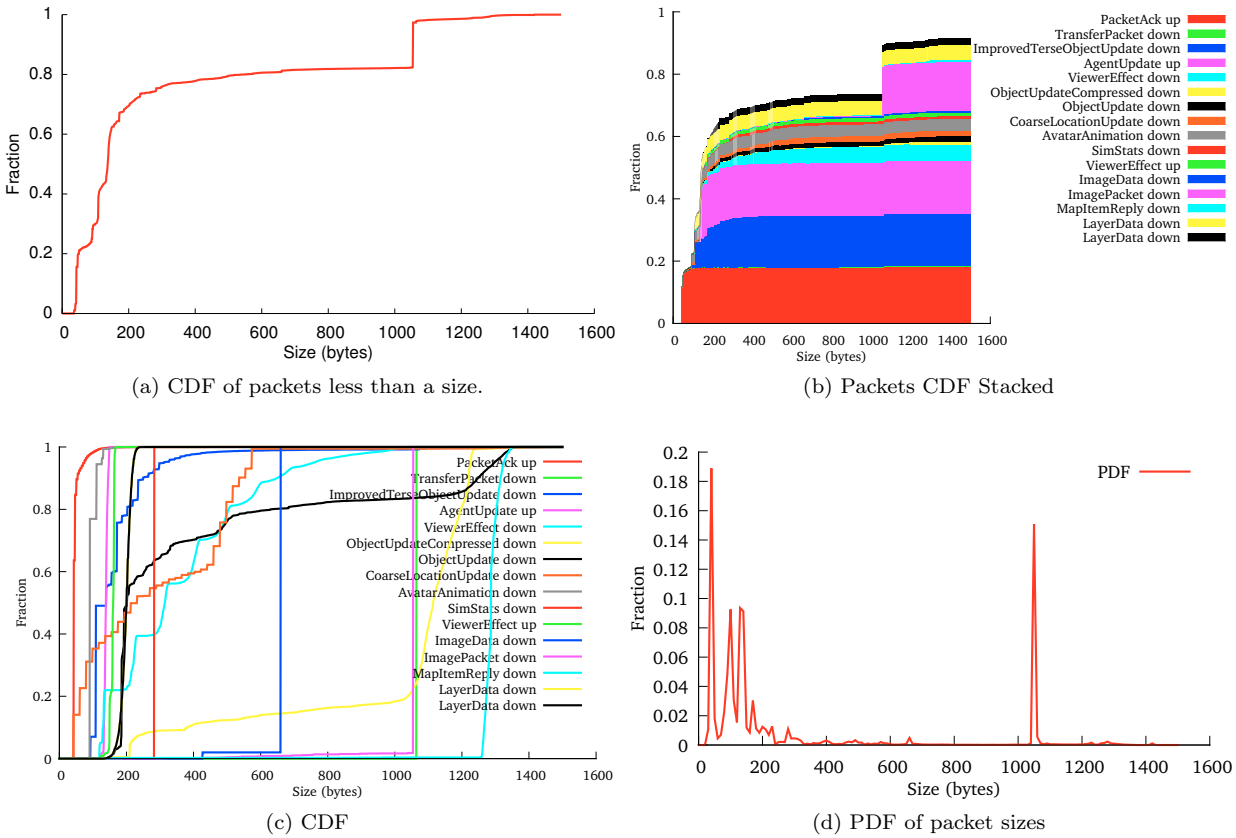
(a) The CDF of packets that are of a size or smaller is shown in figure 5.6a. 69% of the packets are less than 200 bytes in size. 51% of the bytes are in packets 1055 bytes in size, this is 15% of the total packets.

(b) The CDF for the packets that make up more than 2% of the total bandwidth, stacked on top of one another are shown in figure 5.6b. The size of each part represent the portion of the packets that are of that type and that size or smaller. It shows that most of the packets are of one of these 7 types.

(c) The non stacked version for the same packets is shown in figure 5.6c.

(d) The number of packets of each size for the entire connection is shown in figure 5.6d, it shows that a large number of the packets are less than 200 bytes.

The CDF of packet sizes for machine 1 are shown in figure 5.7. The x-axis shows the packet size in bytes and the y-axis shows the fraction. The fraction of the packets that are of certain size or smaller are shown in figure 5.7a. The



(a) CDF of packets less than a size.

(b) Packets CDF Stacked

(c) CDF

(d) PDF of packet sizes

Figure 5.6: The distribution of packet sizes.

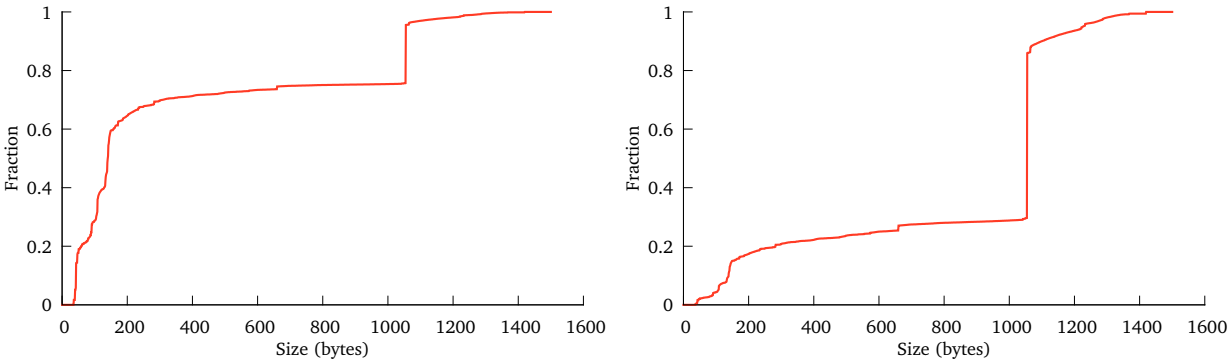
fraction of the data that is in packets of a size or smaller is shown in figure 5.7b. This machine has less small packet packets than the entire trace.

Information about the different packet type present in the trace is shown in table D.1 in appendix D. This is the types of packet in both directions. The downstream Task packets are mostly different type of ObjectUpdate, the different versions save space by reducing the amount of unnecessary information that they include. There are only 4 types of packets in the Texture channel, most of which are ImagePacket packets. The different ObjectUpdate packets can apply to all of the different types of Object. The different types of Object are Prim, Avatar, Grass, NewTree, ParticleSystem and Tree.

5.2.6 Circuits

The per machine statistics for all of the machines during the session are shown in table D.2, in appendix D. There are more machines than the number of users, three of the machines have no Second Life circuits so they never actually logged onto Second Life. There is a wide variation in the number of different connections, there is also a variation in the amount of bandwidth used. The machines with more connections don't all have more bandwidth usage. This is despite each participant being asked to complete a similar set of activities. The ratio between TCP and Second Life Circuit traffic remains broadly similar. There are a few anomalous machines 4, 26 and 34 were not used and so have no bandwidth usage. Discounting these machines the average bandwidth utilization per machine for the entire trace varies between 84 and 168 Kbps with the modal value being 120 Kbps.

There is however considerable variation of bandwidth usage over time. To illustrate this we look in more detail at the traffic of machine 3. This machines traffic is similar to the average, it had slightly more than the modal amount of traffic. Over the lifetime of the session 195 megabytes of data were transferred, giving a mean transfer rate of 128 Kbps. This is significantly below the bandwidth cap imposed by the Second Life throttle, which was set to 500 Kbps and should allow up to 750 Kbps.



(a) Proportion of the packet less than this size.

(b) Proportion of bytes that are in packets of that size or smaller.

Figure 5.7: The distribution of packet sizes for machine 1.

5.2.7 Channels

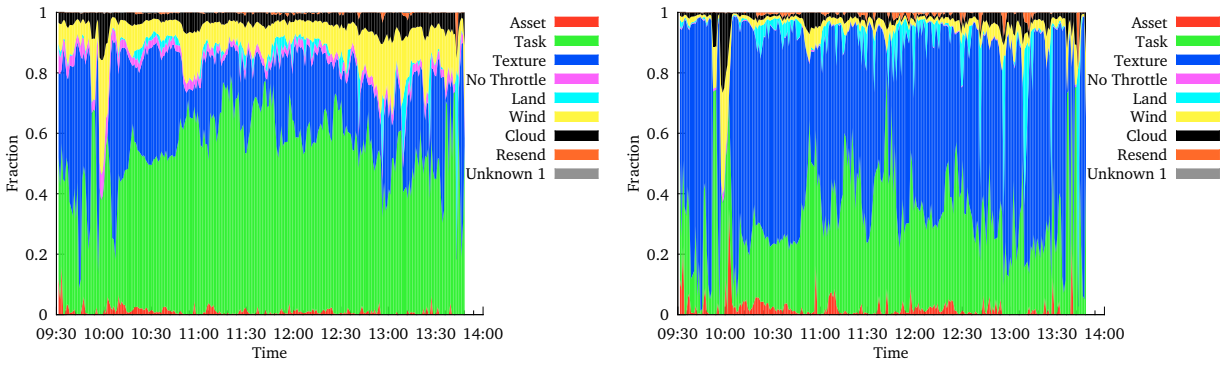
Second Life divides the packets flowing within a circuit into different channels. As discussed in subsection 4.3.5 the total bandwidth for a circuit is divided between the different channels. It is not a straightforward task to measure the flow of data across channels because Second Life packets do not have a channel identifier. Instead each packet type is allocated to a channel, so if a mapping has been established between packets and channels it is possible from packet traces to decompose a circuits flow into its component channels. This has been done by inspecting the OpenSim server code and relevant documentation. Hence the analysis program is able to identify which channel each packet belongs too.

The summary statistics for the channels are shown in table 5.5. The same information averaged by the number of machines are shown in table 5.6. It has been compiled by analysing all of the traces and then calculating an average value for each machine. The first column gives the name of each channel. The second column gives the number of packets received, column three the number of bytes received. Column four gives the average packet size for the channel, column five labelled bps gives the throughput in bits per second. Column six labelled throughput gives the measured percentage of the total throughput that belongs to that channel. Column seven labelled allocated gives the percentage of the circuits throughput that is allocated to that channel by Second Life. There are a number of points of interest:

- The analysis program is able to allocate 99.7% of packets to a channel.
- A large number of the packets and bytes are in the Texture and Task channels. Most of the bytes are from the Texture and Task channels. The Texture packets are mostly large and the Task packets are mostly small. This is not surprising as the texture channel mainly carries information about how to paint the environment. The

Total	**Number**	**Bytes**	**Average Size**	**Throughput (bps)**	**Percentage of Throughput**
NoThrottle	3,140,537	149,311,585	47.54	9,313.35	3.08%
Asset	122,895	100,628,511	818.82	6,276.73	2.08%
Task	8,406,389	1,645,966,839	195.80	102,667.59	33.96%
Texture	2,604,490	2,624,811,154	1,007.80	163,723.25	54.15%
Land	198,321	158,445,980	798.94	9,883.11	3.27%
Wind	719,027	79,277,633	110.26	4,944.96	1.64%
Cloud	306,605	60,995,889	198.94	3,804.63	1.26%
Resend	25,092	26,214,716	1,044.74	1,635.15	0.54%
Unknown	13,475	1,487,459	110.39	92.78	0.03%
Total	15,536,831	4,847,139,766	311.98	302,341.55	100.00%

Table 5.5: Summary statistics for the channels.



(a) Packets (b) Bytes

Figure 5.8: The bandwidth used by each of the channel throttles.

task channel amongst other things contains information about an avatars movements.

- There are also a large number of packets that are not in a channel these packets are very small in size.
- The task channel accounts for some 33% of all bandwidth. This is significantly more than is required for Avatar movement and is accounted for as it is used as a hold all for packets that don't fit elsewhere.
- The Texture channel accounts for some 54% of the total bandwidth.
- There is a discrepancy between the bandwidth allocated by Second Life to channels' throttles and that utilized by the channels, Task using marginally more and Texture significantly more. Meanwhile all the others use less bandwidth than is allocated. In the case of land this can be explained as Land will require significant amounts of bandwidth at the start of a session, but once an avatar is established within its location few updates will be required. Thus the long term average will be below the allocated amount.

The summary statistics for the channels for machine 1 are shown in table 5.7. A large number of the packets and bytes are in the Texture and Task channels. There are more bytes in the Texture channel and less in the Task channel than the average. None of the bytes in this connection were re-sent.

The amount of bandwidth used for each channel and how it changes over time are shown in figure 5.8. The values for the number of packet are shown in figure 5.8a and the number of bytes are shown in figure 5.8b. The Texture and Task channel make-up most of the packets and bytes. With the Task channel taking most of the packets and the Texture channel taking most of the bytes.

The under use of channels such as Cloud and Wind and the mixing up of Avatar movement packets with other packet types both suggest that a rethink of the channel categorization would be appropriate. Nonetheless the division of a circuits packets into channels means the basis for applying differential QoS to different packet types within a

Total	**Number**	**Bytes**	**PktSize**	**bps**	**Throughput**	**Allocated**
NoThrottle	101,307	4,816,502	47.54	300.43	3.08%	0%
Asset	3,964	3,246,081	818.82	202.48	2.08%	14%
Task	271,174	53,095,704	195.80	3,311.86	33.96%	29.7%
Texture	84,016	84,671,328	1,007.80	5,281.40	54.15%	29.7%
Land	6,397	5,111,161	798.94	318.81	3.27%	11.3%
Wind	23,194	2,557,343	110.26	159.52	1.64%	2%
Cloud	9,890	1,967,609	198.94	122.73	1.26%	2%
Resend	809	845,636	1,044.74	52.75	0.54%	10%
Unknown	435	47,983	110.39	2.99	0.03%	
Total	501,188	156,359,347	311.98	9,752.95	100.00%	100%

Table 5.6: Summary statistics for the channels average over the number of machines.

circuit exists. In furtherance of this, a client which enables a user to manually set the individual channel throttle levels has been implemented with the view to further experimentation.

The amount of bandwidth and packets in each channel for where the packets are marked as reliable and where the packets are not marked as reliable, referred to as unreliable, are shown in figure 5.9. The top graphs are the reliable packets and the bottom the unreliable. The left hand graphs are per packet and the right hand graphs are per byte. The re-sent bytes are all reliable as expected. Texture and Land packets are reliable. The Cloud, Wind and unthrottled packets are unreliable. There is a large number of Task packets in both and most of the Asset packets are reliable.

5.2.8 Summary

In this section we have shown that:

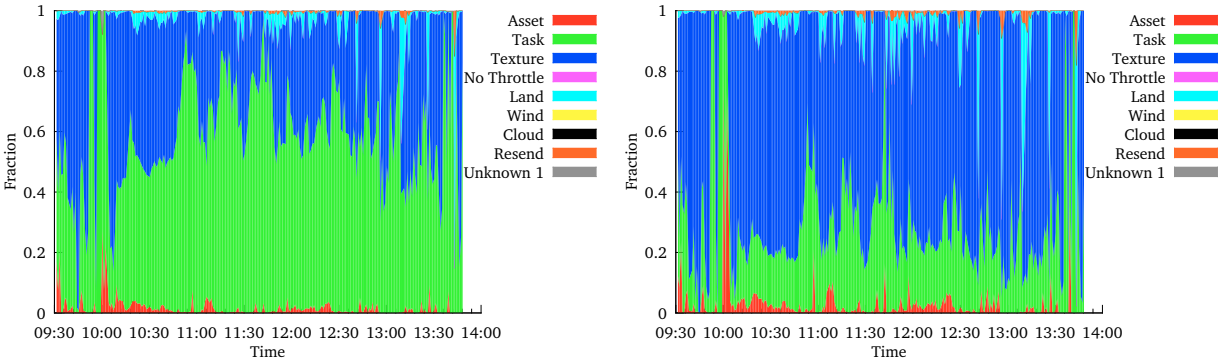
- It was possible to have a lab full of machines simultaneously using Second Life without causing congestion.
- During normal use of Second Life, on an uncongested network, the default settings of Second Life result in it using less bandwidth than it would be allocated using a TCP Fair allocation mechanism.
- When packet loss did occur it was in short lived bursts. Second Life's rate based system is not designed to and was not responsive to these bursts.
- The distribution of packet sizes is such that only a small proportion are full segments. This should be considered when estimating TCP Fair behaviour.
- The task channel is full of small packets, the texture channel is mainly composed of full segment sized packets.

5.2.9 Conclusion

The amount of bandwidth used is far smaller than the amount of locally available bandwidth and the number of losses over the entire lab session is small, so the amount of congestion was small. If the connections were using a congestion control system then they would have been able to increase their bandwidth utilisation. The current system allows the user to adjust the amount of bandwidth used, in a certain range. This requires the user to be able to estimate the amount of bandwidth that they have available for Second Life.

Total	**Number**	**Bytes**	**Average Size**	**Throughput (bps)**	**Percentage of Throughput**
NoThrottle	7,109	301,601	42.43	37.47	0.36%
Asset	3,703	3,309,129	893.63	411.12	4.00%
Task	53,562	17,531,636	327.31	2,178.11	21.21%
Texture	56,888	58,178,781	1,022.69	7,228.08	70.39%
Land	798	476,047	596.55	59.14	0.58%
Wind	14,608	1,613,603	110.46	200.47	1.95%
Cloud	6,233	1,231,372	197.56	152.98	1.49%
Resend	0	0	nan	0.00	0.00%
Unknown	71	14,512	204.39	1.80	0.02%
Total	142,972	82,656,681	578.13	10,269.19	100.00%

Table 5.7: Summary statistics for the channels for machine 1.



(a) Reliable packets

(b) Reliable bytes

(c) Unreliable packets

(d) Unreliable bytes

Figure 5.9: The bandwidth used by each of the throttles.



(a) Minerva (b) Miramare

Figure 5.10: Object creation in Second Life.

5.3 Experimental Investigation

In order to examine other situations the traffic of a single user performing different tasks was recorded. This adds to the understanding of Second Life traffic.

In the thesis statement differential allocation of resources amongst channels is advocated. The traffic of Second Life is separated into different channels that have different functions for the application. This decomposition does not necessarily match the network requirements of the traffic. Analysis of Second Life suggests that its traffic can be decomposed into different traffic types that have different requirements from the network. In particular that the Avatar control traffic is similar in character to network game traffic with respect to timeliness constraints, bandwidth requirements, packet size and inter packet arrival times. Conversely texture traffic, has less stringent timeliness constraints, larger packet sizes and greater variation in bandwidth requirements. This experiment aims to test the above hypothesis.

In addition to examining the different types of traffic the affects of different in-world activities and the effect of this has on the proportion of different types of traffic has been investigated and how this is affected by different settings. In order to investigate the effect of adjusting specific parameters, e.g. throttle setting, and the relationship between traffic and in-world activities such as teleporting, a number of scenarios were designed where controlled activities with specific parameter settings were carried out. In each case the communication with the SL servers hosted by Linden Labs occurred across the Internet against the background of real traffic. A series of experiments was conducted to validate previous studies and to investigate the effect of altering system and user settings. Five sets of experiments were conducted to:

- validate previous studies, client bandwidth and packet size distributions for Standing, Walking, Flying and Teleporting (SWFT) in quiet and occupied islands were investigated.
- establish the accuracy with which the client represents an avatars position, different values for latency, activity undertaken and composition of environment were measured.
- investigate the actual proportion of traffic which belongs to each channel, and the distribution of packet sizes for each channel. Further decomposition to establish the throughput, and inter-arrival times of avatar traffic was undertaken.
- measure the effect of caching by looking at how traffic patterns change as data is built up in the cache.
- establish the relationship between throttle setting and the actual throughput used for the range of global throttle values.

5.3.1 Methodology

The methodology was to conduct a series of controlled experiments, where the avatar performs a series of defined actions within different Second Life scenarios. The traffic that was received and sent from the Second Life client was captured for post processing. Each scenario lasted for ten minutes, and the traffic for each ten seconds will be considered separately, giving sixty samples for each scenario.

Previous work has used scenarios for investigating Second Life. These are areas with large number of objects and avatars and areas with a small number of objects and avatars, areas with a large number of avatars and a small number of objects and areas with small number of avatars and a large number of objects. The case with many avatars and few objects rarely occurs in Second Life as users do not congregate in areas with nothing to interest them.

The experiments were carried out from the University of St Andrews School of Computer Science. Whilst the network connections here are high bandwidth, previous work [53, 97, 102] suggests that a broadband connection would not introduce traffic limitation. So the results would hold for most home users. Mobile broadband, dial up connectivity and a lab full of users may raise issues not dealt with by these experiments.

Traffic statistics were then generated for the global traffic and for individual traffic categories of interest. The results are presented tabularly and through time and distribution graphs.

Two Islands were used for the experiments. One Island (Minerva) was unoccupied and had a total of 5586 objects. The other, Miramare, had an average of 12 avatars present and 8749 objects. The client's cache was emptied between each run. Screen shots of the two islands are shown in figure 5.10, Minerva is shown in 5.10a and Miramare is shown in 5.10b. These islands were selected to investigate the effect of different numbers of object and textures.

We have created a modified version of the Second Life client that outputs the location of the avatar reported by the client and by the server and the difference between those two values. This allows the difference between where the client and server think that the avatar is to be analysed. Walking inside a complex environment can involve lots of changes in direction which will require the faster response from the server.

The behaviour of Second Life with different amounts of bandwidth has been investigated. The amount of bandwidth that Second Life tries to use has been adjusted using the bult-in throttle system. The amount of bandwidth available to Second Life has also been adjusted by using a network level traffic shaping system.

When the throttle (section 4.3.5) total is set to 50 kbps and the user logs in very little of the environment has been downloaded. The land tiles for the island are mostly there, the appearance of the user's avatar may not be entirely downloaded by the time the login is finished and will appear as a white cloud until the download completes. The island appears empty of objects as information about them has not yet been sent to the client. The objects present on the island will very slowly appear in the client. The avatar will react slowly to commands but will be controllable, though the inability to see objects makes navigating around the island difficult. If the amount of bandwidth available is reduced to 75 kbps using traffic shaping whilst the throttle value is left at a large value the environment will appear differently than if the throttle had been adjusted. The avatar's appearance may or may not be finished downloading. The land will not be finished downloading so when the avatar appears it will appear to be standing on nothing. More of the objects will be visible when the avatar appears and more will appear more quickly than if the throttle had been reduced. The avatar will be more difficult to control as it reacts more slowly to commands. This difference is likely to be partly due to the fact that with the throttle total set to a low value the Land channel's throttle gets a larger percentage of the available bandwidth than when the throttle total is set to a large value.

The time that packets take to cross the network affects the responsiveness of the system. With round trip times between 127 and 157 msec the system is usable. When the round trip time increases the system becomes less responsive. When the round trip time is increased by 50 msec to around 200 msec the system remains usable without difficulty. When the round trip time is increased by 100 msec, to 250 msec, the responsiveness of the system reduces making it more difficult to control. At around 450 msec the system becomes difficult to use.

A modified Second Life client was created which is able to determine positional error. Both the client side position and server position were logged, this allowed the discrepancy between the two to be derived and analysed. This was achieved by modifying the function in the program that handles incoming `ObjectUpdate` packets. This function handles all of the different types of packet that contain update information about objects in the environment including avatars. At the part were the avatars positional information is updated to the values from the server, the difference between value from the server and value at the client is calculated and output to the log.

The Second Life client's log is filtered using grep to extract the avatar position information from the modified client. This was processed with a program created for this purpose, this program generates graphs of the error in the position of the avatar.

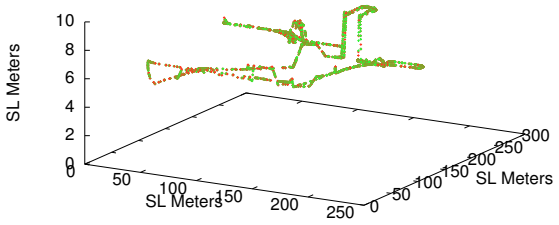


Figure 5.11: The path taken by the avatar

Connections are fed through a router, which was equipped with the netem network emulator traffic [73]. This facilitates control over available bandwidth, levels and pattern of loss, delay and jitter.

In each area a range of activities were undertaken. These activities are walking, flying, standing still and teleporting. The different combinations of activities and areas were used. During the experiments the network conditions were measured to provide context for the work. This gives a total of eight different scenarios that were investigated.

The avatar was put through a sequence of moves. The avatar walked fixed courses in two different islands and flew around the same route, the avatar was also left standing in the same spot for 10 minutes. Two islands were used, both had a number of objects and running scripts. Minerva Island had 5586 objects and no other avatars. Miramare had 8749 objects and other avatars. The cache was emptied between each run. The avatars movements when walking around Miramare island are shown in figure 5.11.

TCPDump was used to capture the packets. A program we designed specifically for analysing Second Life traffic is used in the post processing.

The following metrics were considered for each scenario:

- bandwidth: both the average and the distributions of bandwidth utilisation were investigated.
- packet size: packet size is important with respect to the load placed on the network and deriving models of SL traffic.
- inter-arrival time: this enables comparison of the pattern of network traffic with established patterns for networked games.

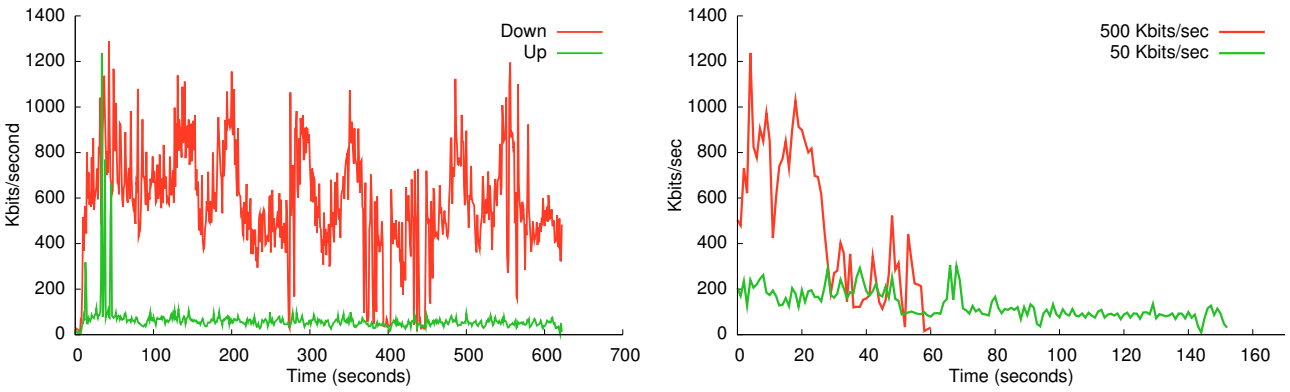
Presentation of Results

Summary statistics are presented. As the distributions are not expected to be normal, robust statistics will be used. Looking at percentiles and medians rather than standard deviation and means. In addition for interesting cases distribution functions, cumulative and probability will be presented.

5.3.2 SWFT: Standing Walking, Flying, Teleporting

These experiments compare SL traffic for the activities of SWFT on a quiet and an occupied Island. First consider a single avatar walking on the occupied Island. The variation in throughput for one run walking on the occupied island, each reading being the average for ten seconds are shown in figure 5.12a. This was for a 10 minute trace. The peaks reach over 1 Mbps and the troughs below 50 Kbps. The throughput from the server to the client is over ten times that from the client to the server. As the avatar progresses around the Island different densities of prims and textures are downloaded, consequently the bandwidth required to support this downloading is also variable. The time series for a teleport is shown in figure 5.12b.

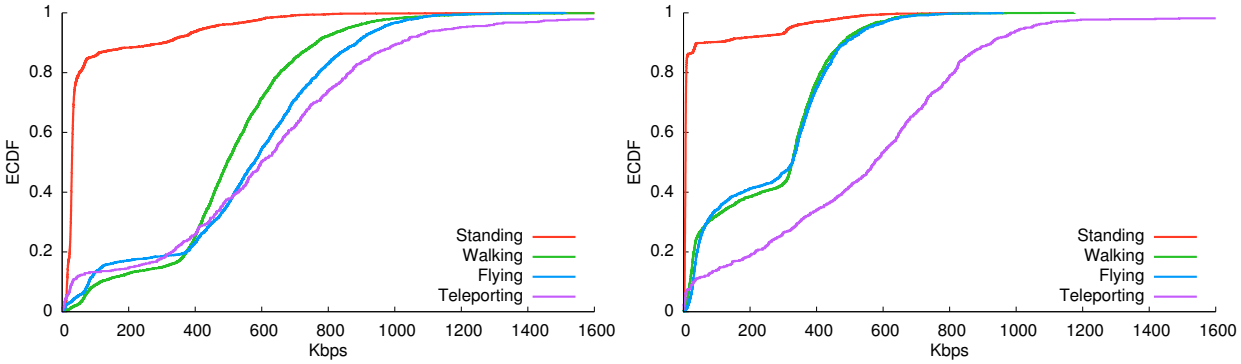
Average throughputs for SWFT for different islands are show in table 5.8, the first two are from [53], the next two are from [6], the next three are from [97] and the last two are from the islands looked at in this study. The average bandwidth utilisation for our measurements ranges from 55 Kbps for standing in on Minerva to 604 Kbps for teleporting. UDP traffic is over 90% of the total traffic.



(a) Flying.

(b) Teleport Delay for Throttles

Figure 5.12: Throughput Against Time



(a) ECDF of Throughput for an Occupied Place

(b) ECDF of Throughput for a Quiet Place

Figure 5.13: The ECDF of throughput for the two islands.

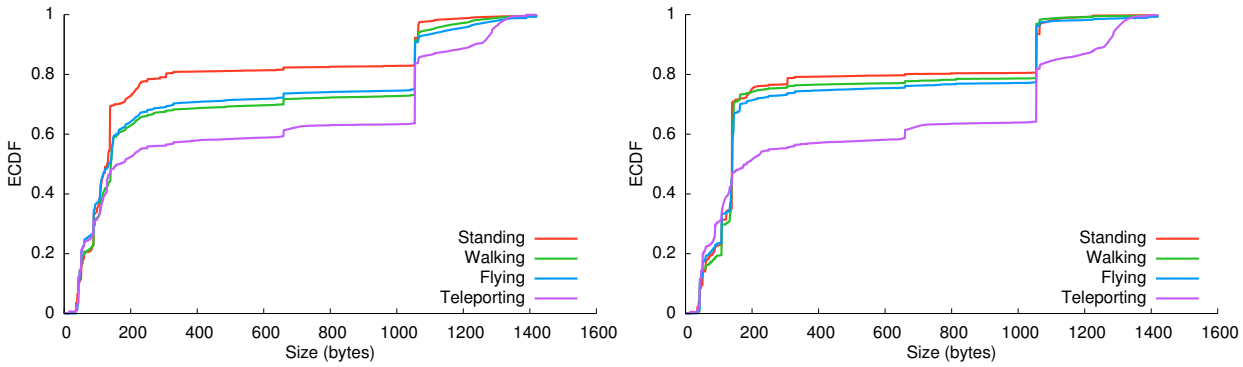
The Empirical Cumulative Distribution Function (ECDF) for the occupied Island, Miramare, is shown in figure 5.13a. The throughput rises from its lowest for standing, through walking and then to flying. The throughput for teleporting reaches a higher peak than any of the other activities, but has periods of slightly lower throughput than flying and walking.

The distribution of packet sizes for SWFT in the occupied Island are shown in figure 5.14. In each case most of the packets are smaller than 150 bytes and there are few packets between this size and 1100 bytes. For teleporting over 40% of packets are larger than 1100 bytes, for walking or flying around 30% are larger whereas for standing less than 20% are large packets.

These throughput values for walking on Miramare are greater than the average for the hands-on session: 466 Kbps compared to 231 Kbps. This is accounted for as the hands-on session would have included a mix of activities, with the avatar standing and chatting or figuring out what to do next.

The results confirm that the throughput of SL varies considerably with time and is orders of magnitude greater than that for 3D games. The throughput required increases with the number of avatars and with the number of objects in a location. Furthermore, the activity being undertaken effects throughput, with a rising rank order of standing, walking and flying. For the environment these experiments were conducted in, it remains a fraction of the TCP Fair value.

These results are in line with that from previous studies. Whilst higher levels of throughput have been recorded, this was for an island with more prims and avatars than the one used here. Previous work has observed that the throughput required for MUVEs like SL is orders of magnitude greater than for computer games like First person shooters (FPSs), Massively multiplayer online role-playing games (MMORPGs) and Real Time Strategy games, which require between 6 and 10 Kbps [52].



(a) ECDF of Packet Size for an Occupied Place

(b) ECDF of Packet Size for a Quiet Place

Figure 5.14: The ECDF of Packet Size for the two islands.

5.3.3 Representation of Avatar's Position

The accuracy with which the system can represent an avatar's position is important for the usability of the system. As latency increases discrepancies are introduced between the user's, client's and server's estimation of where an avatar should be. When the discrepancy grows large it diminishes the fidelity of the Virtual World and the accuracy with which tasks can be completed. In this section we present measurements of the discrepancy between server and client estimations of the Avatar's location.

A scatter plot of the discrepancy between the client and server's estimation of avatar position in the XY plane is show in figure 5.15a. The graph is for an average Round Trip Time of 150 ms and for clarity omits a few outliers. 99% of readings have an error of under half a meter. This is sufficient error to enable avatars to appear to partially walk through walls or to interfere with other activities requiring precision, such as a paintball game. However, it allows most activities that would be undertaken in SL to proceed smoothly.

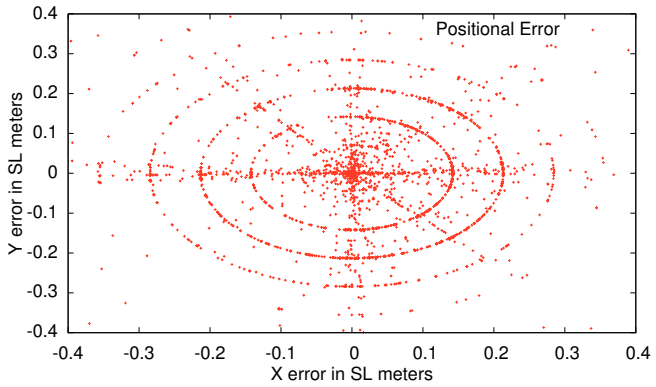
The concentric circles visible in 5.15a occur where the avatar is moving on the flat without obstruction and the client believes it is stationary, or the client believes there is movement and it is stationary. The concentration of readings in the X and Y direction are a result of the avatar's pattern of movement.

The empirical probability distribution function of the absolute error in three dimensions for RTTs of 150, 300 and 750 ms is shown in figure 5.15b. The distributions extend beyond the x-axis at low concentrations. At all latencies error is concentrated in several regularly spaced peaks. This corresponds to the concentric circle discussed above and reflects the error that can accumulate between packets from the server to the client.

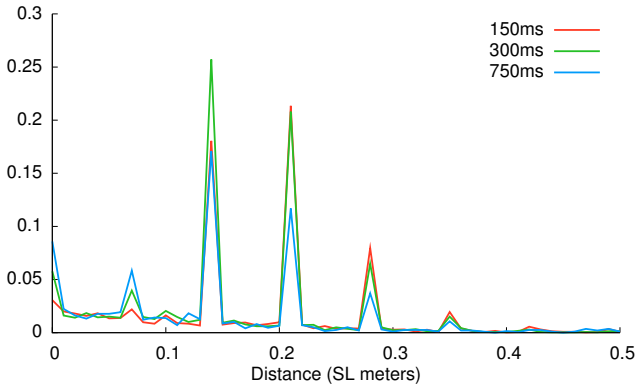
The error at the 75th, 90th, 95th, 99th and 99.9th percentiles, for Round Trip Times ranging from 150 ms to 750 ms is shown in figure 5.15c. There is a significant deterioration in usability over these timescales, with the simple task of walking round the Island becoming difficult to complete at RTTs above 500 ms. There is no measurable increase in error below the 90th percentile. This suggests that error in a small proportion of samples has a disproportionate effect on usability. Note that the X axis is to a log scale and the increase in error for the 95th, 99th and 99.9th percentile is

	Downstream (Kbits/s)				
SL island	**Country**	**St**	**W**	**F**	**T**
Goddess of Love	BZ	150	240	220	-
Magdelen	BZ	25	70	75	-
Hippie pay	BZ	255	330	250	-
Tuskany IV	BZ	20	77	75	-
Isis	US	192	703	1,164	877
Cyclops	US	141	278	445	821
Solaris	US	10	31	448	27
Minerava	UK	55	295	349	604
Miramare	UK	102	466	589	528

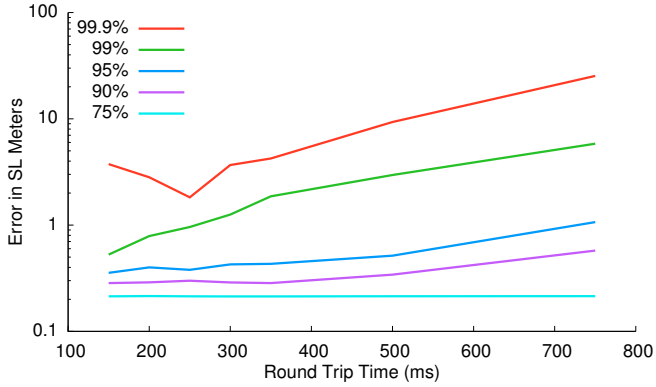
Table 5.8: Throughput for SWFT on different Islands.



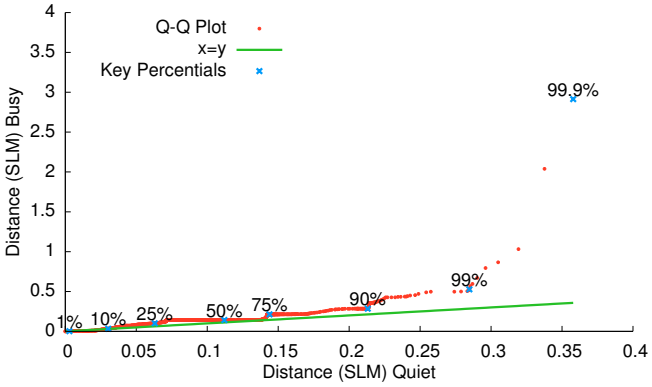
(a) Positional Error in the XY Plane



(b) Distribution of Positional Error



(c) Percentile Error at a Range of Latencies.



(d) QQ avatar position error.

Figure 5.15: Positional Error

approximately linear on this scale.

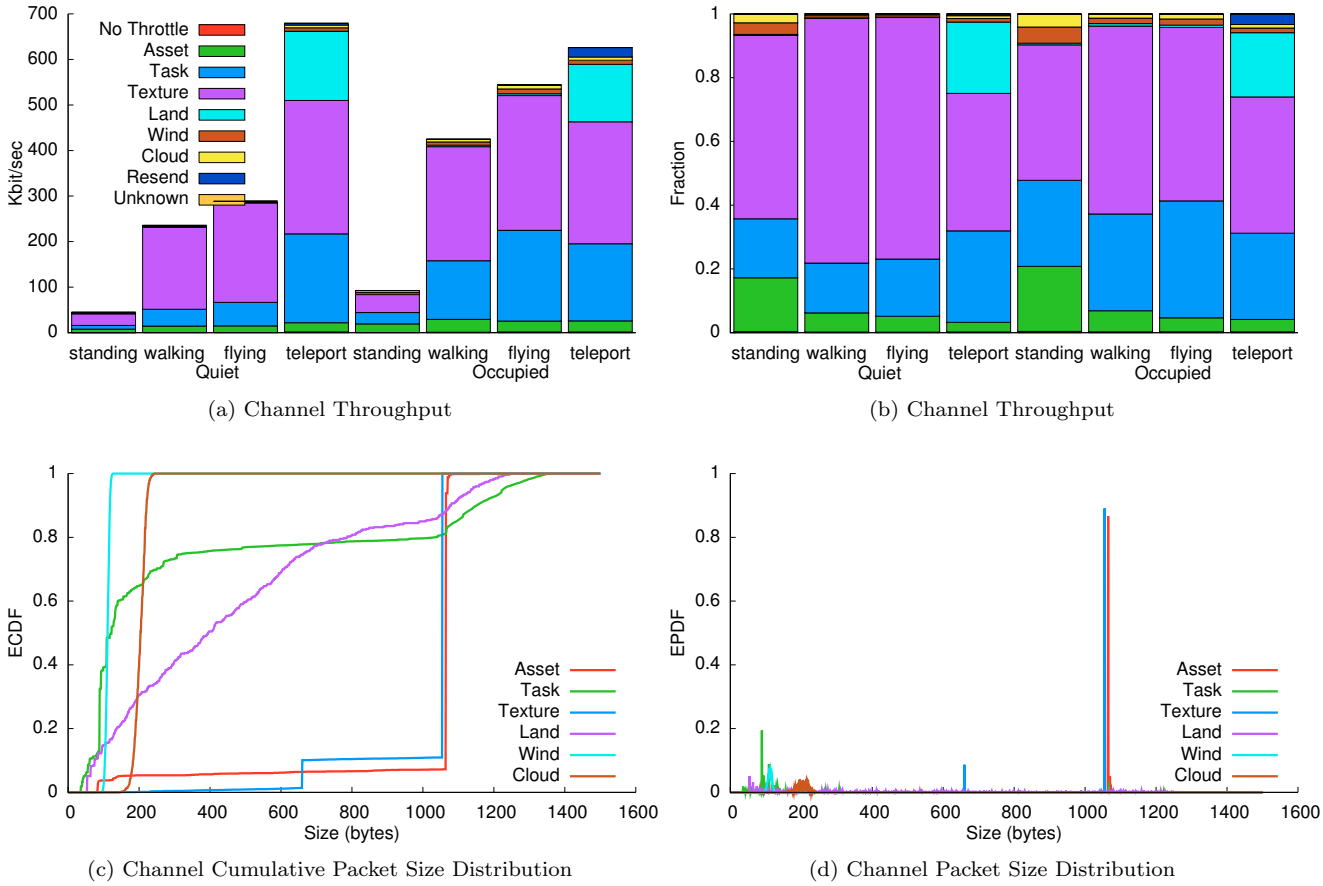
The errors for walking with flying on the occupied and the quiet Island are compared in table 5.9. The error for flying is systematically higher than that for walking. At the 90th percentile it is five times greater on the quiet Island and four times greater on the occupied Island. The error is systematically larger for the occupied Island. For walking and flying it is almost twice that on the quiet Island as is shown in the table. The quartile-quartile plot of position error for walking on the busy and quiet is shown in figure 5.15d. The x-axis shows the position error in SLM for the quiet island and the y-axis shows the position error in SLM.

The measurements presented here show that the size of the error depends upon latency, the activity being undertaken and upon how busy the environment is. At 150 ms RTT the error in avatar representation is within the range where it has an impact on the usability of the system. Consequently, it is preferable to control, and where possible reduce, this error.

Changes in the throttle value affect the amount of packets relating to the user's avatar that are sent from the server to the client. When there are less packets relating to avatars the difference between the clients estimate of the avatars position and position sent by the server increases. The decrease in the number of packets per unit time increase the

Scenario	Error in SL meters for given percentile					
	50	75	90	95	99	99.9
Walking Quiet	0.10	0.14	0.21	0.21	0.25	0.33
Walking Occupied	0.18	0.21	0.28	0.36	0.53	3.75
Flying Quiet	0.62	0.71	1.05	1.07	1.41	3
Flying Occupied	1.03	1.39	1.77	2.10	3.5	8.21

Table 5.9: Error in SL meters at 150 ms RTT



(a) Channel Throughput

(b) Channel Throughput

(c) Channel Cumulative Packet Size Distribution

(d) Channel Packet Size Distribution

Figure 5.16: Decomposition by Channel

inter arrival time for those packets. This gives the client and server more time to differ in their estimation of the avatars position. As the round trip time increasing increase the time between the packets being sent from the server to the client and also the commands being sent from the client to the server that affect the simulation. Increase in the round trip time therefore increase the positional error in the avatar's position.

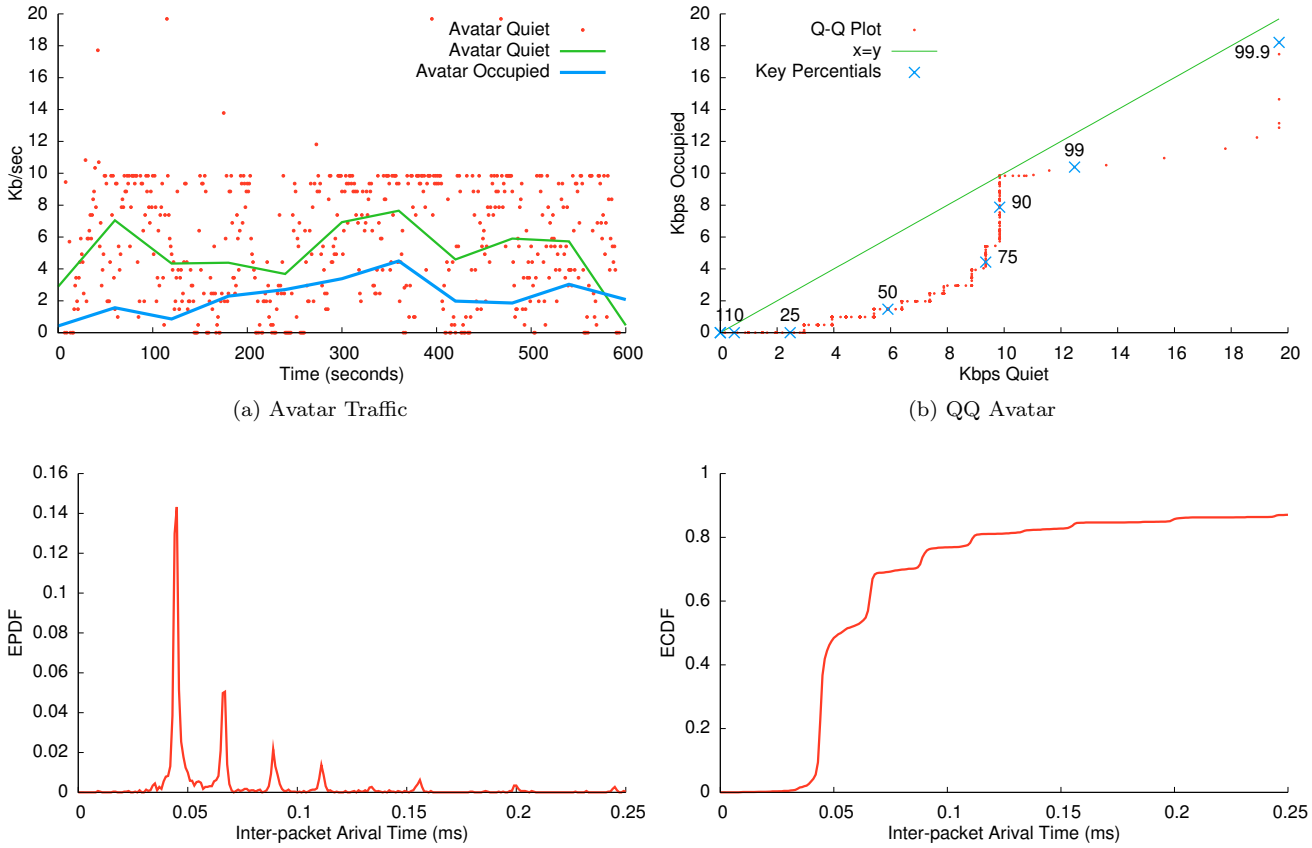
5.3.4 Decomposition by Channel

SL both places a cap on total throughput and allocates bandwidth between eight channels. Allocation and utilisation may however differ as the demand for bandwidth varies with avatar activity. This section presents and discusses measurements for each of the channels, for SWFT in the quiet and the occupied Island. The user setting for the global throttle is set to its default of 500 Kbps. Bandwidth, packets per second and the distribution of packet sizes are examined in turn.

The decomposition by channel of throughput, by channel for SWFT on the quiet and occupied Island is shown in figure 5.16a. For standing, walking and flying the largest component comes from the texture channel, with the asset channel second. For teleporting there is a significant land component. This is in part because the land for the source and destination of the teleport is included. There is also a small amount of traffic in the asset channel. Cloud and wind have very little traffic. The resend channel is negligible for each of the cases except for teleporting to the occupied Island. The level of resend required depends upon network conditions, particularly congestion. This suggests that teleporting stresses the network more than other activities.

The wide distribution of packet sizes for each of the channels are shown in figure 5.16c. The Asset channel has 90% of its traffic with packet sizes over 1000 bytes; the Texture channel has a similar distribution. The Task channel is mostly made up of small packets, 60% of them are less than 150 bytes. The Land texture has a wide spread of packet sizes approximately evenly distributed over the range from 40 to 1400 bytes. The remaining Wind and Cloud channels have very little variation in packet size with averages of 112 and 204 bytes respectively.

The distributions of channel packet sizes vary little for Standing, Walking, Flying and Teleporting or between the



(a) Avatar Traffic

(b) QQ Avatar

(c) Avatar Inter Arrival Time

(d) Avatar Inter Arrival Time CDF.

Figure 5.17: Avatar Traffic

different islands, however there is significant change in the overall packet size distributions. This suggests that changes in the proportion of traffic belonging to each channel is important in determining the packet size distribution and consequently should be taken into account when modelling MUVE traffic.

The small packet sizes for the task channel means that although most data bytes belong to the Texture channel most of the packets belong to the Task channel.

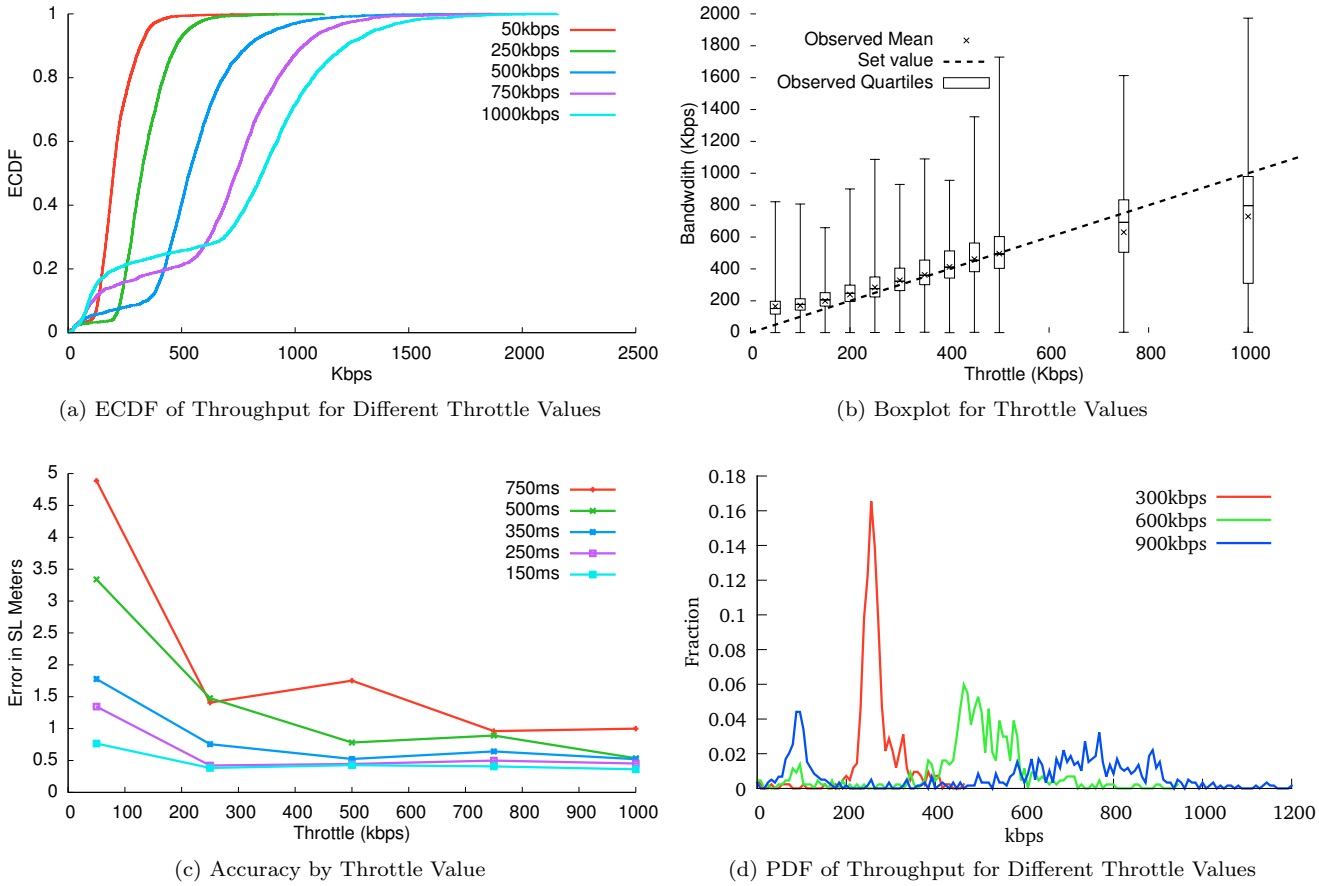
5.3.5 The Task Channel and Avatar Control

The Task channel contains packets which update: prims, avatar control, the particle system, trees and grass. We look at the composition of the Task channel for Walking on the occupied and quiet Islands. The total bandwidth for the Task channel is 110 Kbps on the occupied Island and around 20 Kbps on the quiet Island. In both cases traffic for trees, grass and the particle system is negligible. Avatar control traffic averages less than 10 Kbps and the majority of traffic is for prims.

The bandwidth used by avatar control traffic is shown in figure 5.17a. The scatter plot gives the average throughput for a second. Each dot corresponds to the throughput averaged over a second. The line graphs give the throughput average for each minute. With the exception of a few outliers the maximum throughput in each second is 10 Kbps. This is a similar to the total throughput for 3D games. The quartile quartile graph of the bandwidth of avatar traffic for the occupied island and the unoccupied island is show in figure 5.17b.

The amount of avatar traffic for unoccupied places is higher than for occupied places. This is despite the overall Task traffic for the occupied Island being more than five times that of the quiet Island and the activity undertaken by the avatars being the same on both Islands.

The distribution of inter-arrival times for avatar control traffic is shown in figure 5.17c. Over half the packets have an inter-arrival time of slightly less than 50 ms. There are then regular but decreasing peaks in the distribution up to slightly more than 150 ms. These peaks correspond to the circles in figure 5.15a. The walking speed of an avatar is 3 SLM/sec (Second Life meters per second). The distance travelled by a walking avatar in the time between one peak



(a) ECDF of Throughput for Different Throttle Values

(b) Boxplot for Throttle Values

(c) Accuracy by Throttle Value

(d) PDF of Throughput for Different Throttle Values

Figure 5.18: Throttles

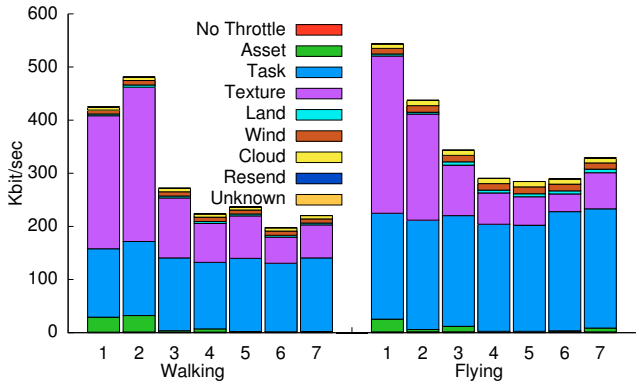
in inter arrival time and the next approximates to the distance between peaks in figure 5.15b. Thus regularities in the distribution of inter-arrival times provides an explanation for regularities in the accuracy of avatar representation.

The reduction in bandwidth for avatar control traffic correlates with a decrease in the accuracy of avatar positional representation. The reduction in avatar traffic for the occupied Island explains the decrease in accuracy of representation. It follows that if it was possible to provide a QoS guarantee for the avatar control traffic then accuracy of avatar representation and consequently the usability of the system would be improved. *This could be achieved through the introduction of an Avatar Control traffic channel.* If the network supported end to end differential service [14] SL would be able to classify Avatar Control Packets as requiring priority service.

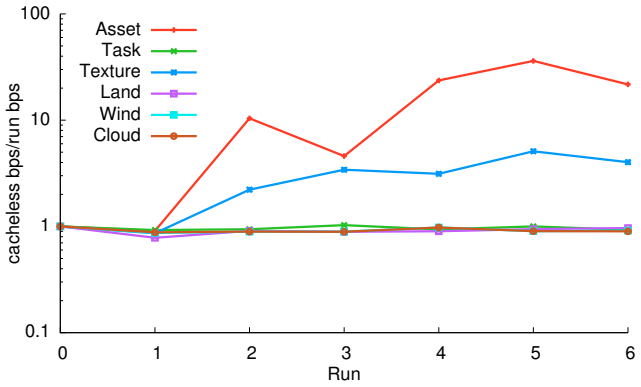
5.3.6 The Second Life Client Cache

SL clients maintain a cache of data between sessions, which can be set by the user to between 10 MB and 1 GB. It defaults to 500 MB. Here the effectiveness of the cache is evaluated by conducting a sequence of scenario runs on the same Island without flushing the cache between the runs. The runs labelled "1" in figure 5.19a were conducted immediately after flushing the cache. The run labelled Walking "2" was done next again with an empty cache. From here the cache was allowed to accumulate. Flying "2" was conducted next, followed by Walking 3, Flying 3 etc. Each of these pairs of runs was carried out on consecutive days. As the cache builds up, the average throughput required falls to about half of that when the cache is empty. This is accounted for mainly in the reduction in bandwidth required by the Texture channel. There is little change in the bandwidth requirement for the Task channel.

The sensitivity of each channel to caching can be determined by comparing the throughput required when the cache is empty to the throughput required in consecutive runs. The ratio of throughput required when the cache is empty to that for consecutive runs as the cache grows is shown for walking on the occupied Island in figure 5.19b. The Asset channel's bandwidth requirements is reduced by over 90% and Texture's by over 75%. The other channels are unaffected.



(a) Sequential Runs, Effect of Caching



(b) Channel Sensitivity to Caching

Figure 5.19: Effect of Caching

After a week of conducting experiments and not flushing the cache, the cache grew to around 157 MB suggesting that the default setting of 500 MB is large enough.

In summary, when a user revisits the same area or Island caching reduces the bandwidth. The Asset and Texture channel are both helped by caching, however the effect on other channels is minimal. The cache was flushed when performing all of the other experiments.

5.3.7 Throttle Settings

The SL client contains a dialog box for network settings, where the user may adjust a slider to set "bandwidth" to any value between 50 Kbps and 1500 Kbps. The default setting is 500 Kbps. In this section we seek to find out what effect, if any, changing the settings on the bandwidth slider have on SL.

At the start of a session and periodically throughout its life the SL client communicates a throttle value to the server. The server uses this value to limit the UDP bandwidth it uses when sending to the client. The throttle value sent to the server is one and a half times the user selected value or 1,500 Kbps whichever is smaller.

In each experiment the avatar was walked around a set course on the occupied Island. Each run lasted ten minutes and throttle settings between 50 Kbps and 1000 Kbps were used. Runs were also conducted where extra latency was introduced giving RTTs of 150 ms to 750 ms. The distribution of bandwidth utilisation, the accuracy of avatar representation and the time it takes for a download to complete after a teleportation are discussed below.

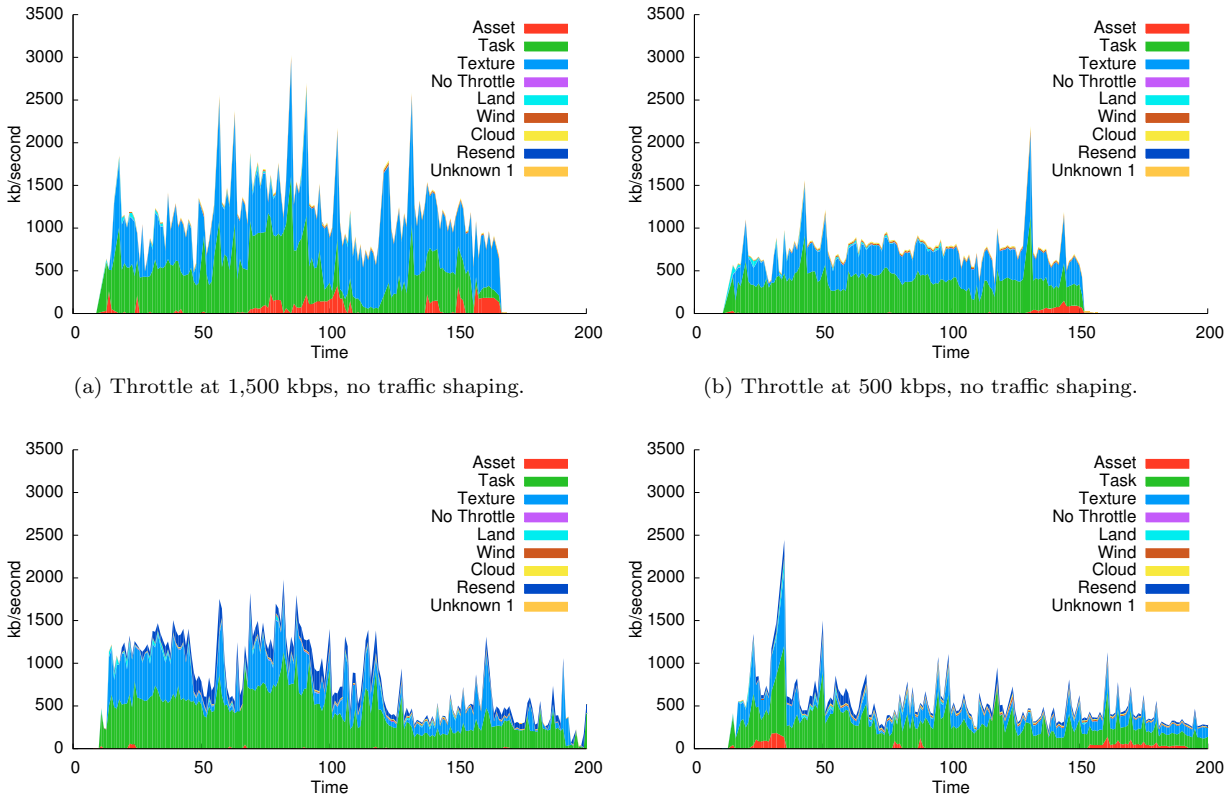
In order to test this hypothesis, we reproduced the activity undertaken in the workshop, but with the settings of the Second Life Host changed to a cap of 300, 600 and 900 Kbps respectively. This experimentation was performed with a single user, on an island with buildings an open space and no other avatars the same receiving rate was observed. The throttle value was then set to different values and the rate at which packets were received change by the expected amount.

The distributions for throttles set to the respective values are shown in figure 5.18d. When the throttle is set to 300 Kbps the distribution is tighter than for the higher throttle values. It peaks at around 280 Kbps. When the throttle is set to 600 the distribution is looser with a peak around 500 and a sharp drop off at 600 Kbps. When the throttle is set to 900 Kbps the distribution is looser still with a peak at just below 800 Kbps and very little traffic above 900 Kbps.

These figures suggest that, these connections are limited by the sending rate, though packets are not being sent at the rate indicated by the AgentThrottle packets that are being sent out. The sending rate under these conditions is equal to the sending rate set by the user. Most of the traffic captured is being throttled by the system. Second Life's throttle system does limit the amount of bandwidth used and adjustments to its settings does result in changing the cap on peak bandwidth usage.

Furthermore the peak bandwidth utilization appears to be around a third of that suggested by the TCP Fair equation discussed in section 2.2.3. This would suggest that measurement based traffic control would have allowed more network traffic and therefore made the system more responsive to users.

Empirical distribution functions for throughput are shown in figure 5.18a. The mean throughput values and the user throttle settings along with box plots of the median, lower quartile, upper quartile, minimum and maximum



(a) Throttle at 1,500 kbps, no traffic shaping.

(b) Throttle at 500 kbps, no traffic shaping.

(c) Throttle at 1,500 kbps, traffic shaped to 500 kbps

(d) Throttle at 500 kbps, traffic shaped to 500 kbps

Figure 5.20: The bandwidth used by each of the throttles under different network conditions.

value, are shown in figure 5.18b. As the throttle level increases so too does the mean throughput, which ranges from 170 Kbps to 780 Kbps. However, the user does not get exactly what they asked for. At low throttle settings mean throughput is above the setting, at high throttles it is less than the setting and at 500 Kbps the user setting and mean coincide. At 50 Kbps throttle the throughput is over three times the setting. The variation in throughput increases with the throttle value.

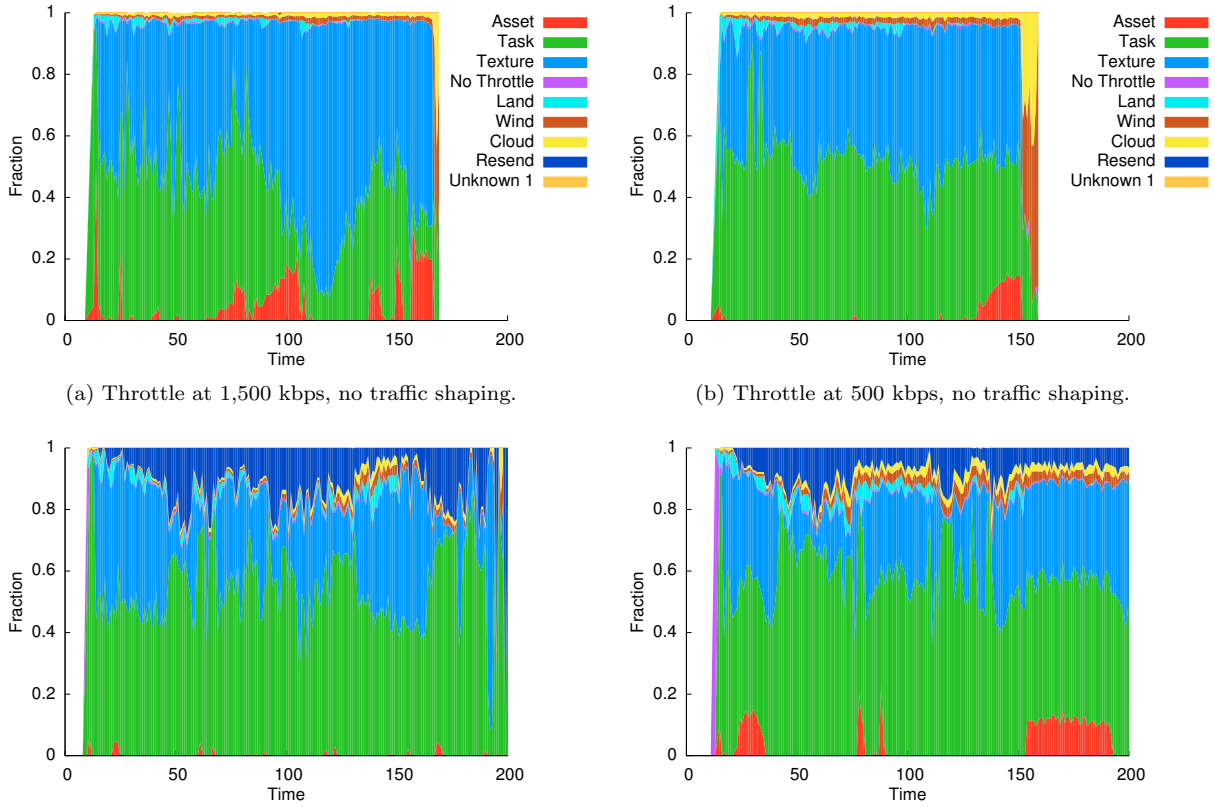
The 95th percentile of error in the client position of the avatar are shown in figure 5.18c. As discussed previously error in a relatively small percentage of samples has an important effect on user experience. This graph shows that avatar representation is more accurate at higher throttle settings and at lower RTTs.

Time series which illustrate the delay between a teleportation starting and all relevant data being downloaded are shown in figure 5.12b. Delay is reduced at the higher throttle level. It is also the case that the time it takes for elements of the environment to download is reduced at higher throttle levels.

The amount of bandwidth used by each channel throttle is shown in figure 5.20a. The percentage of the bytes that are in each channel throttle is shown in figure 5.21a. For a connection where the throttle is set to the maximum and no traffic shaping on the connection. The amount of bandwidth allocated to each channel throttle is shown in figure 5.22a. These traces are shorter than the other traces.

The amount of bandwidth used by each channel throttle is shown in figure 5.20b. The percentage of the bytes that are in each channel throttle is shown in figure 5.21b. For a connection where the throttle is set to 500 kbps and no traffic shaping on the connection. When there is less bandwidth the mix of packet types is different. A greater percentage of the bandwidth is given over to the task throttle. This prioritizes the movement of the avatar over download of the objects in the environment. A larger proportion of the bandwidth is given over to the land throttle so that the land appears before the objects and the avatar. The amount of bandwidth allocated to each channel throttle is shown in figure 5.22b.

The amount of bandwidth used by each channel throttle is shown in figure 5.20c. The percentage of the packets that are in each channel throttle is shown in figure 5.21c. For a connection where the throttle is set to the maximum and the traffic is shaped to 500kbps on the connection. The land throttle gets very little of the bandwidth in this configuration which results in there being gaps in the island as the avatar moves around. The avatar is also less



(a) Throttle at 1,500 kbps, no traffic shaping.

(b) Throttle at 500 kbps, no traffic shaping.

(c) Throttle at 1,500 kbps, traffic shaped to 500 kbps

(d) Throttle at 500 kbps, traffic shaped to 500 kbps

Figure 5.21: The bandwidth used by each of the throttles under different network conditions.

controllable. The amount of bandwidth allocated to each channel throttle is shown in figure 5.22c. In this case the system is not converging to the amount of available bandwidth and the server is slow to react to changes in the value set by the client.

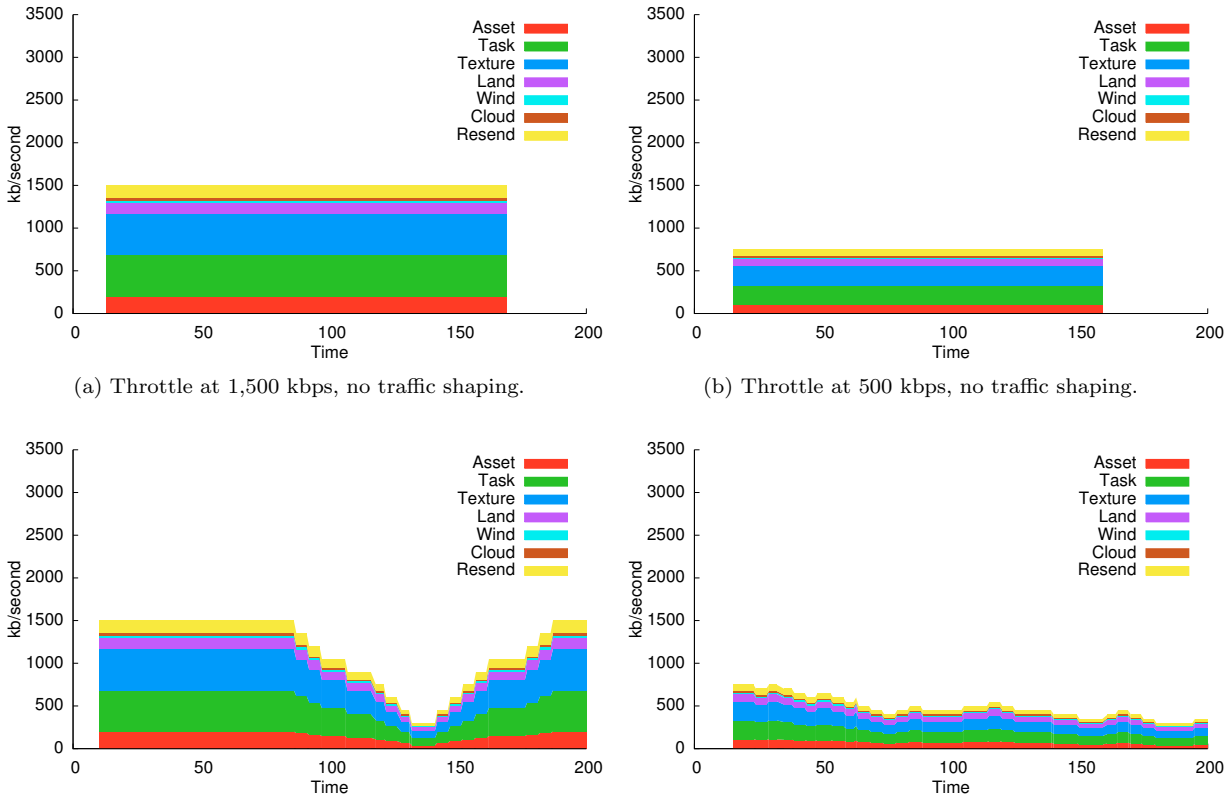
The amount of bandwidth used by each channel throttle is shown in figure 5.20d. The percentage of the packets that are in each channel throttle is shown in figure 5.21d. For a connection where the throttle is at 500kbps and the traffic is shaped to 500kbps on the connection. The task throttle is larger for this configuration and the land throttle is also larger. The results in the land appearing before the other objects and causes the avatar to be more controllable than with the same amount of bandwidth and the throttle set to a larger amount of bandwidth. The amount of bandwidth allocated to each channel throttle is shown in figure 5.22d. In this case the system is coming closer to the correct amount of bandwidth.

When the system is attempting to use the amount of bandwidth available it works better than when it is attempting to use more bandwidth than is available. The current system changes the amount of bandwidth used in increments, indirectly, determined by the user. When the value set by the user is 1500 the increment value is 3 times larger than when a value of 500 is set. An adaptive traffic management system will detect network conditions to control the number of packets transmitted so that it competes fairly with other connections on the network and prevents congestion. Congestion control is important because it prevents congestion collapse and allows network resources to be used fairly and efficiently.

In summary, changing the user bandwidth setting has an important effect upon the bandwidth utilisation. At higher bandwidth settings accuracy was improved and delay reduced. This is not a setting which should be ignored either by end users or those studying Virtual World traffic.

5.3.8 Throttle Adaptation

When connecting to SL, a user may experience a range of network conditions. RTT and level of loss may vary because of factors such as location, time of day and access technology. This is increasingly the case as the range of hand held devices and the popularity of mobile computing increases.



(a) Throttle at 1,500 kbps, no traffic shaping.

(b) Throttle at 500 kbps, no traffic shaping.

(c) Throttle at 1,500 kbps, traffic shaped to 500 kbps

(d) Throttle at 500 kbps, traffic shaped to 500 kbps

Figure 5.22: The bandwidth allocated to each of the throttles under different network conditions.

The throttle adaptation system is described in section 4.3.4. The behaviour of SL under a range of loss conditions was evaluated. Using netem, runs were done with added loss of 0, 1.2, 3 and 6 percent for the Min (50), Max (1500) and default (500) throttle values. The loss rate, throughput and throttle values were logged each second.

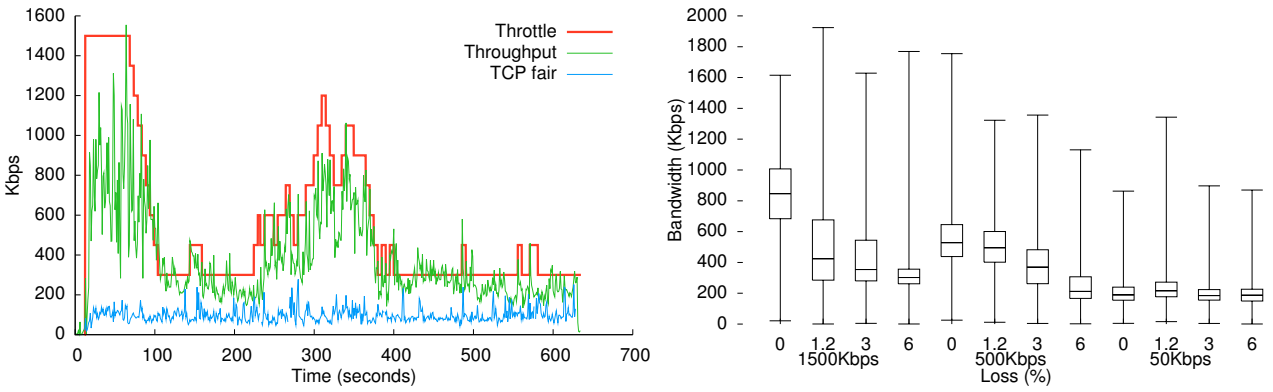
A time series of throttle values, the measured throughput and the TCP Fair bandwidth is shown in figure 5.23a, the mean loss rate is 3%. Two things are striking: firstly, the throughput is effectively capped by the throttle value; secondly, the throttle level increases from 200 to 400 seconds, without there being an identifiable reduction in congestion.

The box plots in figure 5.23b show that a throttle of 1500 adapts more to congestion than other settings. However, throughput is higher than the other settings both at low and high loss levels. When the cap is set to 50 Kbps there is little variation in throughput.

5.3.9 Summary

In this section we have shown that:

- The traffic generated depends upon the scenarios being undertaken, number of avatars, motion and the number of objects in proximity.
- Second Life's throttle system does limit the amount of bandwidth used and adjustments to its settings does result in changing the cap on peak bandwidth usage.
- That Second Life would be able to make use of extra bandwidth allocated to it and in so doing would improve the usability for the system.
- The distribution of packet sizes is such that only a small proportion are full segments. This should be considered when estimating TCP Fair behaviour.
- The task channel is full of small packets, the texture channel is mainly composed of full segment sized packets.



(a) Throttle values and measured throughput.

(b) SL Throttle Trace

Figure 5.23: SL Adaptation to Loss

This leads us to conclude that dynamically adjusting the overall throttle size in response to congestion, would allow Second Life to be more responsive to network conditions. Furthermore much of the time it would allow Second Life to benefit the user by making more use of the bandwidth available.

5.3.10 Conclusion

Second Life traffic can be decomposed into throttles which have distinct characteristics with respect to bandwidth requirements and packet size distributions. Avatar control packet is a small proportion of the total traffic (less than 10 Kbps) but has a disproportionate effect on system usability.

The state of the cache has a significant effect on the amount of texture traffic that is required and consequently on the packet size distribution and total bandwidth requirements. Adjustments in the global throttle effects both the distribution of throughput and the accuracy of avatar representation.

The existing system that Second Life uses to control the rate it transmits packets is not TCP friendly under all circumstances, as it does not react to changes in the round trip time. Under the sorts of network conditions it normally operates Second Life uses less than its fair share of the available bandwidth. The users experience can sometimes be improved by more bandwidth being available to the system. The load on the server can also affect the usability of the system.

5.4 Second Life traffic requirements

Second Life traffic contains different traffic types of traffic. Second Life's traffic is bursty and some of its traffic is not sensitive to timing constraints. The ordering of this traffic is not important. The connections are long lived, should be congestion controlled and are between a client and several servers. If TCP were used for the Texture traffic there are two ways this could be organized, it could be as many small connections, one for each transfer, or as a small number of connections, one for each server. Neither organization of the connections would result in it acting a single TCP fair connection. The large number of connections include a lot of extra overhead and would not reach a steady state. With a long lived connection for each server each connection would be able to take a "TCP fair" share of the network bandwidth resulting in the system taking several times a TCP fair share of the network bandwidth. It would be possible to limit the bandwidth used by the TCP connections using a separate system. TCP would then be providing reliability and ordered byte stream semantics and adding extra overhead. There is an existing system to provide reliability and a packetised system without ordering allows independent data to be used when it arrives rather than being delayed to be put in the correct order.

The avatar traffic requires a small portion of the available bandwidth but is more important. It has similar properties to the traffic of game like environments. It is sensitive to loss and delay, contains small packets and is necessary for the interactivity of the system. For this reason it would be best if the traffic control system had a lower bound on the allowed bandwidth utilization. The allocation of the channel throttles should ensure that the amount of bandwidth allocated for the avatar movement is sufficient for its requirements. It would be preferable that if when

packets from a Second Life circuit were dropped that they were from the Texture channel rather than being avatar packets.

The traffic generated by MUVEs is application limited for, a possibly large, part of the time. The TCP fair equation does not take into account application limited periods, it instead assumes that the traffic will always be network limited. The amount of bandwidth the congestion control system believes that it can use may not be accurate. A window based system will have increased its window during all of the time that it was application limited to be much larger than the amount of bandwidth used and possibly much larger than the amount of bandwidth available. A rate based system, on a network with assumed properties, will have reached the level permitted by the round trip time and packet loss rate but will not have been able to take into account limits on the total bandwidth. Using the calculated value could result in the system using more bandwidth than the network can handle. It would therefore make sense for the system to maintain a known value for the largest amount of bandwidth that it is known can be used on the network.

These requirements would be best served by a network that was capable of providing different services to different flows of packets. Explicit Congestion Notification (ECN) [163] provides the possibility of packets being marked rather the dropped to indicate that the network is congested, this would be desirable for all of the traffic but especially the traffic regarding the avatar movement. Diffserv [14] allows traffic to be marked so that the routers treat it differently. This would allow the avatar traffic to be given preferential treatment and allow the texture traffic to be dropped instead of it when there was congestion. Asynchronous Transfer Mode (ATM) [38] also provides the possibility of asking the network to treat different traffic differently.

Second life currently uses UDP for most of its traffic which provides no congestion control but allows it access to packet out-of-order as they arrive. TCP would provide congestion control where the packet transmission rate changes rapidly and is designed to provide the maximum throughput and would impose ordering, stream based semantics and reliability. Second Life creates several connections between the same hosts, this would suggest that there is opportunity to share information about the state of the network between the different connections. DCCP is a transport level protocol that provides congestion control and is connection oriented. It dos not provide a stream abstraction, ordering or reliability which Second Life does not need. As Second Life uses unicast for its UDP packets so it is possible for it to use a connection oriented protocol without changing the communication architecture that it uses. DCCP was designed to allow the addition of new congestion control protocols, its current protocols do not provide the desired properties.

The requirements of Second Life traffic management are: timely delivery of packets, this is not provided by TCP as it delays data delivery to the application to provide in order delivery. It requires up to 200 Kbps per circuit required throughput. It must adapt to network conditions by changing the rate of packet transmission. An adaptive buffer can be used to reduce jitter. It must also provide sufficient bandwidth to make the MUVE usable, or when this is impossible inform the application. It should share information between the connections that share the same route over the Internet. It should not exceed its fair share of the available bandwidth.

The traffic management system will control the rate at which packets are sent onto the network. Whilst there are other factors which affect the QoS provided, these cannot be directly affected at the end points of the connection. Packet loss is often an indication of congestion on the network, so a system that controls the rate at which packets are sent out onto the network will be able to reduce the rate of packet loss and balance the advantage of higher bandwidth with disadvantage of a higher packet loss rate. The delay in the network is also affected by the packet transmission rate as when more packets are sent the number of packets in the queues in the routers increases and results in packets waiting longer to be transmitted. The affect of a single connection reducing its transmission rate will depend on the number of other connections going through the same route and queueing systems used by the routers. Jitter can be affected directly by adding extra delay at the end points, if expected arrival time of the packet is known. In systems where packets are transmitted at a consistent rate and are useless after the time when there would have been used, it is some times possible to delay the use of all of the packets so that jitter causes less of the packets to be discarded.

SCTP [179] like DCCP is connection oriented and provides packet rather the stream semantics. It provides both ordered and unordered deliver of data. SCTP is ordinarily reliable though there exists an extension to send packets unreliably. SCTP has a congestion control system however it is designed to have single congestion control system and is not designed to support alternatives. RTP used with RTP Control Protocol (RTCP) used to provide feed back to an application about the network conditions and this can be used to create a system to control the packet transmission rate based on network conditions. The RTCP packet provide feedback on network conditions, these are separate packets it must be keep down to 5% of the RTP packets. This arrangement restricts the congestion control system to being a formula based congestion control system. The use of RTP to provide this type management would put it in the application layer of the protocol.

5.5 Summary

Second Life traffic has been examined in two different ways. Second Life traffic was captured during a lab, the lab was to introduce users to Second Life for educational purposes. The traffic was passively captured by mirroring the traffic to a port on the routers and capturing the traffic using TCPdump on a computer attached to the port. The captured traffic was examined using a program created for this purpose, which is described in chapter 3.

This measurement lead us to a number of conclusions. That it was possible to have a lab full of people using Second Life and the normal settings in a lab do not cause congestion problems. The traffic depends on what is happening in-world in the vicinity on the user's avatar. The packets are not full sized and differ in average size for the different channels is different. Second Life is limited by its throttle system and would be able to use more bandwidth if the throttle adjustment system set a higher value. This leads use to conclude that a congestion control system would allow Second Life adapt to network conditions and use more bandwidth.

An experimental investigation of Second Life has been performed where an avatar was put through. The throttle was decomposed into channels and the distinct traffic characteristics. The avatar traffic takes up a small amount of bandwidth. The cache was investigated and discovered to have an effect on the amount bandwidth used.

The existing Second Life traffic management is not TCP fair under a wide range of network conditions. It does not react to changes in the RTT as it does not take into account the RTT in calculating the rate.

Chapter 6

Adaptive traffic management for MUVEs

The previous chapter has shown that Multi User Virtual Environments (MUVEs) make considerable demands upon the network paths that interconnect clients and servers. For example, in order to allow an avatar to move smoothly around an island, Second Life (SL) will often require more than ten times the bandwidth of a comparable Massively Multiplayer Online Role Playing Game (MMORPG), in a similar scenario. It was also shown that there is a considerable range of bandwidths over which Second Life could function. At the lower end of the scale (around 50 to 100 kbps) the avatar was difficult to control and scenes presented slowly. At the higher end of the scale (up to about 1.5 Mbps) the avatar's control is smoother and even complex scenes are presented quickly. Whilst the application would function at moderate bandwidth, performance would improve at higher bandwidth.

From the users' perspective it is clearly better to have a smoothly running system. However, one consequence of the ubiquity of the Internet has been a growth in the variety of access technologies. At one end of the spectrum users may log on from a lab that connects to the Internet at Gigabit speeds at the other end of the spectrum connectivity of tens of kilobits per second may be provided through a mobile network using General Packet Radio Service (GPRS). Even when there is good connectivity if the path is being shared between many users any individual's fair share may be a small fraction of the physical provision.

Consequently, it makes sense to have an adaptive system, which can make use of bandwidth when available but will continue to operate under constrained network conditions. The system should give Second Life a fair share of the available bandwidth, so that it can make optimal use of the network without freezing out other users' traffic. The Transmission Control Protocol (TCP) is responsible for carrying over 80% of Internet traffic [61, 28, 89]. Whilst not perfect TCP's congestion control mechanism allows a form of statistical fairness between competing streams. The behaviour of TCP is therefore an important reference point for the design of new protocols. If it can be shown that Second Life traffic can be shaped to behave in a TCP fair manner it follows that such traffic is unlikely to pose a threat to the stability of the network. This chapter describes the design, implementation and evaluation of a TCP fair adaptive traffic management system.

The system described is for Second Life which in 2010 is the virtual world with the largest user base. Whilst Second Life is a proprietary system run by Linden labs the client is open source. This chapter will describe the design of a modified Second Life client which is compatible with the existing Second Life system servers, this client will be called Mongoose. This in turn will mean that Mongoose is compatible with the important open source virtual world OpenSim.

Section 6.1: Second Life and TCP traffic management systems are compared with each other.

Section 6.2: The design space for adaptive traffic management systems is described. The constraints on the system by Second Life are described, then the design of the system is given.

Section 6.3: The structure and data flow of the client and the implementation of the new adaptive traffic management system is described.

Section 6.4: Summarises the chapter and describes the output of the system.

6.1 Comparison of SL and TCP adaptive traffic management

In order to evaluate the appropriateness of current Second Life traffic management this section will present a comparison of TCP and Second Life. First the pertinent parts of the discussion in chapter 2 are recapped in order to establish what is meant by TCP fair behaviour. The behaviour of the TCP fair equation is compared with measurements of TCP traffic, allowing us to validate and establish the accuracy of the formula we will be comparing Second Life against. Next, a concise description of Second Life's adaptive traffic management system is presented. We then proceed to a high level comparison of the TCP and Second Life congestion control algorithms. Under conditions of deterministic loss, across a range of round trip times and loss regimes, the question of how the TCP and Second Life algorithms compare is addressed. This is important as it establishes in what ways, if any, it would be beneficial to modify Second Life's traffic management system. This modified traffic management system would involve a modified client to modify the behaviour of the server traffic limiting system.

6.1.1 TCP

TCP uses packet loss as an indication of congestion. The packets have sequence numbers and when a packet is not acknowledged for a length of time after it has been sent it is considered to have been lost. If a receiving host receives a packet out of sequence it will send an acknowledgement of the last in-order packet for every out-of-order packet it receives. The sender will take this as indication of loss when it has received 3 duplicate acknowledgements. TCP uses a congestion window (CWND) to limit the number of in-flight packets. This window can grow in two different ways; slow start and congestion avoidance. A variable called SSTHRESH is maintained to determine which. When the CWND is less than SSTHRESH it is in slow start and window size increases by one packet for each acknowledgement. When CWND is more than SSTHRESH it is in congestion avoidance and the CWND increases by one packet size per round trip time (RTT). The two types of loss detection are treated differently, duplicate acknowledgements cause the CWND and SSTHRESH to be set to half the CWND, a timeout causes SSTHRESH to be set to half CWND and CWND to be set to two packets. The majority of losses are detected with duplicate acknowledgements, this results in the CWND increasing to level and then being halved and then increasing back to that level. This is the stable equilibrium state of TCP.

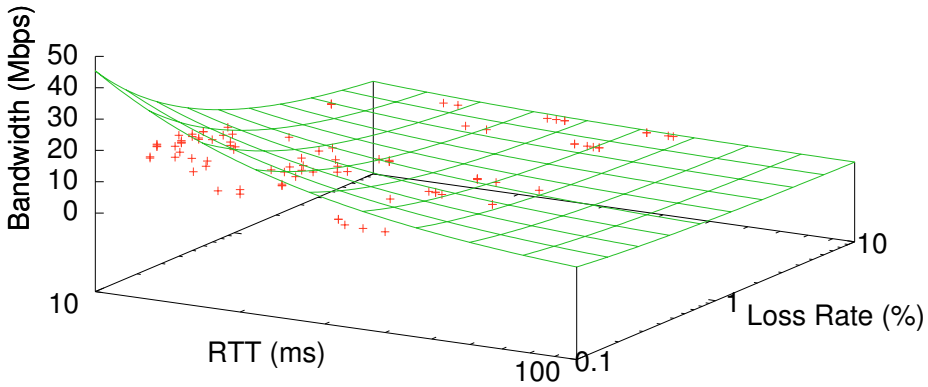
$$X = \frac{C * S}{RTT\sqrt{p}} \tag{6.1}$$

TCP's congestion control system is additive increase, multiplicative decrease (AIMD). Systems with this property will converge whereas some other types of system will not [27]. The long term bandwidth utilization of TCP is given by the formula 6.1. Where S is the maximum segment size, RTT is the round trip time, p is probability of loss and C is a constant. A congestion control system that uses less than or equal to this amount can be said to be TCP fair.

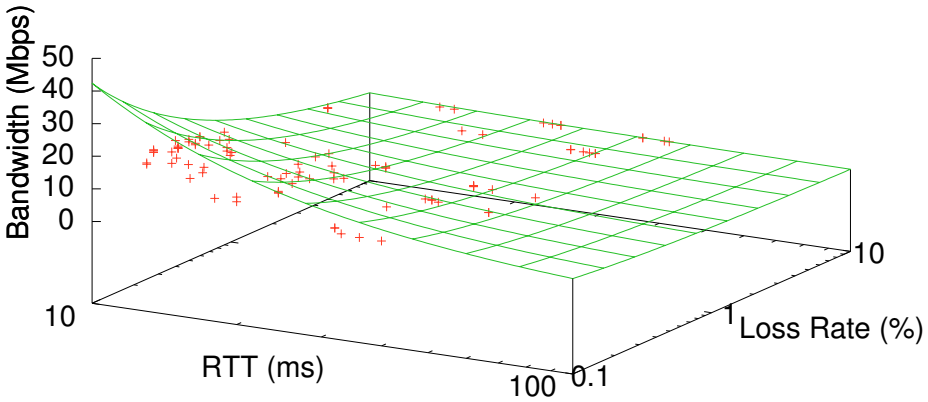
6.1.2 Validating the TCP fair equation

To get information about the reaction of TCP to network conditions a test network where the round trip time and loss rate could be adjusted was used. Different round trip times and loss rates were set on the network. A kprobe [101] module was used to extract information from the kernel about the round trip time and congestion events detected by the TCP implementation. The congestion control system for TCP was set to Reno. The congestion event information, packets size and amount of data transferred was used to calculate a loss rate to ensure that the loss matched the set loss rate. The extra delay and loss was introduced using a different router for each direction. User Datagram Protocol (UDP) packets were sent over the test network to determine if the change in sending rate was a result of the congestion control system in TCP or processing limits in the network. The rate at which packets and bytes were being received at the receiving machine was recorded. The UDP packets were sent in two different fashions, first they were sent without any limits on the rate at which they were sent this lead to a receiving rate a little less than 2 MB/s. The second method was to limit the rate at which packets were sent to 40 Mbit/s, which is 5 MB/s, this value gave the largest receiving rate. The receiving rate was 5 MB/s when no loss was introduced and when the loss increased to 10% the receiving rate reduced to 4.5 MB/s, which is 10% less, the receiving rate recoded was not affected by the round trip time. The traffic that the network is capable of handling appears not to be affected introducing loss and delay other than the change that is the result of the loss or delay.

The results with the TCP friendly equation and the Reno version of the equation are shown in figure 6.1. The x-axis shows the RTT in milliseconds, the y-axis shows the percentage loss rate and the z-axis shows the bandwidth in



(a) A graph of the loss rate, round trip time and transfer rate for TCP connections and for the TCP rate equation.



(b) A graph of the loss rate, round trip time and transfer rate for TCP connections and for the Reno version of the TCP rate equation.

Figure 6.1: 3D graphs of TCP fair rate equation and measurements.

Mbps. The points are the samples from the test network, using the average values. The green lines are the equation plotted for the range of values.

(a) The plot of the simple version of the TCP fair equation is shown in figure 6.1a.

(b) The plot of the Reno version of the TCP fair equation is show in figure 6.1b.

The samples from the test network shown in 2D plots are shows in figure 6.2. The points are the samples from the test network. The lines are the TCP fair equation. In all the graphs y-axis are the bandwidth in Mbps.

(a) The transmission rate against the round trip time for connections with a loss rate 0.1% are shown in figure 6.2a. The x-axis shows the RTT in milliseconds.

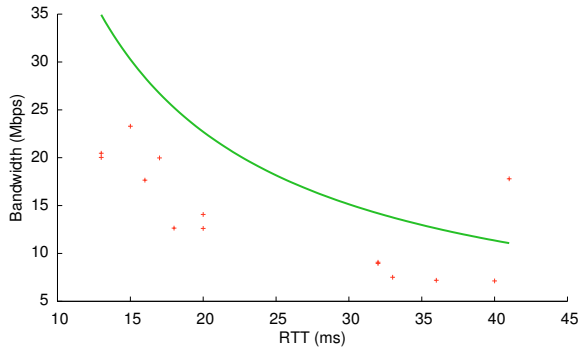
(b) The transmission rate against the loss for the connections, with round trip times between 10 ms and 20 ms are shown in figure 6.2b. The x-axis shows the percentage loss rate.

(c) The transmission rate against the round trip time for a 1% loss rate is shown in figure 6.2c.

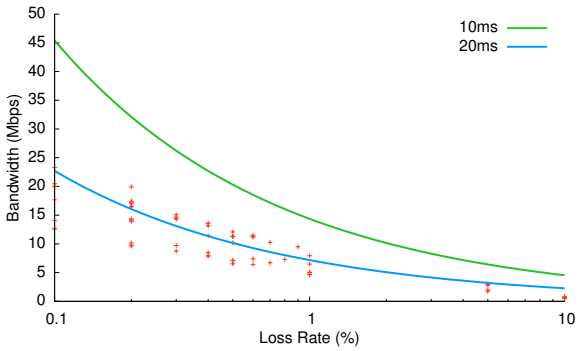
These values are similar to the values in [146] which gives the Reno version of the equation. The values are closer to the Reno formula than the more general equation. This replicates that experimentation, confirming that results comparable.

6.1.3 Second Life traffic management

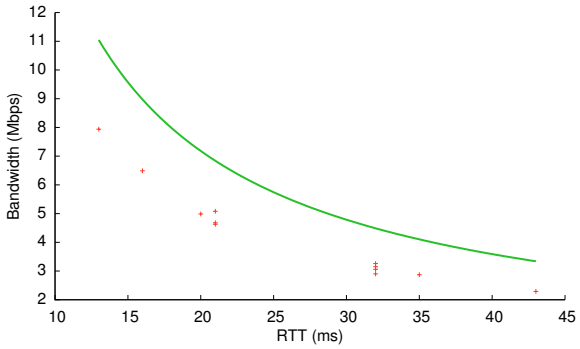
When connecting to SL, a user may experience a range of network conditions. RTT and level of loss may vary because of factors such as location, time of day and access technology. This is increasingly the case as the range of hand held devices and the popularity of mobile computing increases.



(a) The round trip time against the transfer rate for TCP connections with a loss rate of 0.1%.



(b) The loss rate against the transfer rate for TCP connections with a round trip between 10 ms and 20 ms.



(c) A graph of the round trip time and transfer rate for TCP connections with a loss rate of 1%.

Figure 6.2: 2D graphs of TCP equation and measurements.

The source code of the Second Life client was examined and from the following was determined. Second Life tracks the level of network congestion and adapts its bandwidth utilisation. Initially the cap on bandwidth may be set by the user to any value between 50 and 1500 Kbps. The initial value used is the smaller of 1500 Kbps and 1.5 times the user setting. During the session the bandwidth cap may range between 0.1 and 1.5 times the initial user setting.

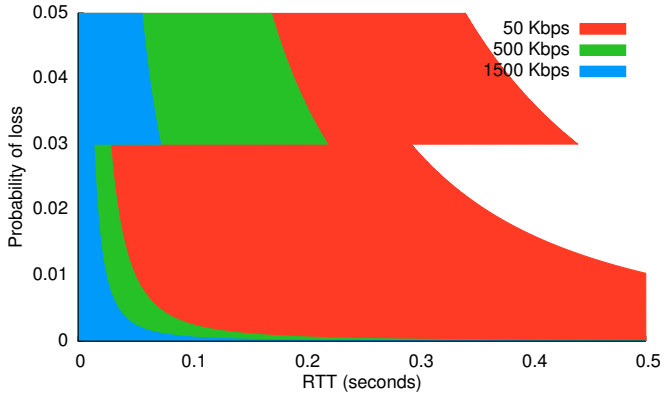
- When the loss rate is less than 0.5% and the current cap is less than the starting value the rate is increased by 0.1 times the set rate.
- When the loss rate is greater than 3% the sending rate is reduced by 0.1 times the set rate.

This is an additive increase and an additive decrease in transmission rate, which will not lead to convergence in transmission rates sharing the same network path [164]. A loss rate larger than 3% will not cause the transmission rate to increase at a faster rate. The transmission rate is independent of round trip time. This is a rate based formula based system that does not include the RTT in the calculation of the rate. This system therefore does not react to the RTT directly. These properties of the congestion control system are not consistent with TCP and are not TCP fair.

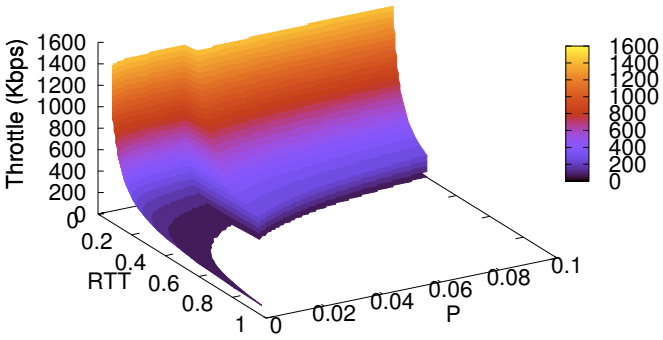
The loss detection system used by Second Life is timeout based. The packets have sequence numbers which are used to detect gaps in the sequence space of packets that are possibly lost. The RTT calculation is used to create a timeout time for packets that are possibly lost. This is used to calculate the percentage of packets that are lost.

6.1.4 Comparison of SL and TCP fairness

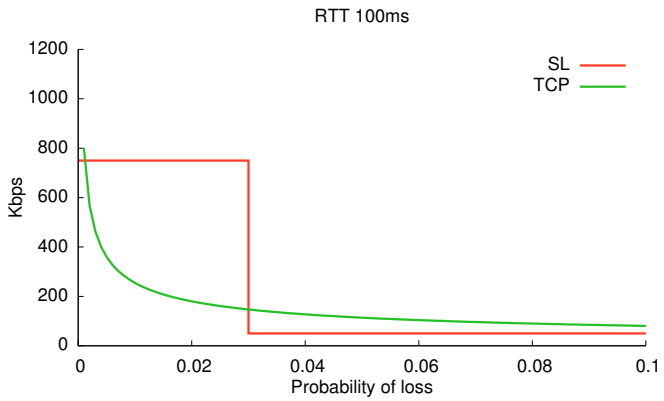
Under the assumptions that the data source is not application limited, loss is deterministic and for long lived flows the mean bandwidth produced by the TCP Fair equation and SL adaptation algorithm can be trivially calculated. For RTTs up to 500 ms and for loss regimes up to 5% whether the TCP or SL algorithm will generate more bandwidth is shown in Figure 6.3a. The x-axis is the RTT in seconds and the y-axis shows the probability of loss. Three shaded areas are shown for the minimum, default and maximum user bandwidth settings. The leftmost shaded area represents the



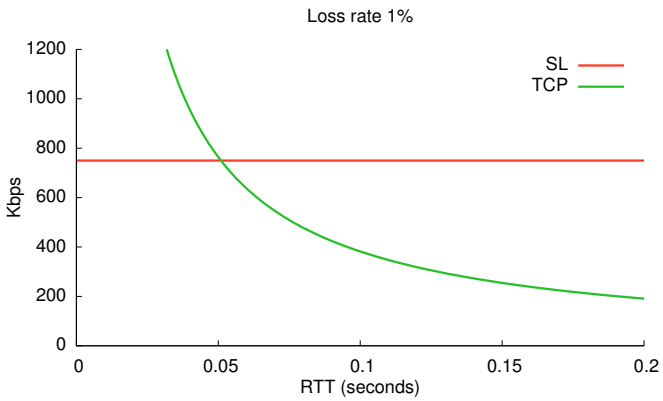
(a) TCP and SL Comparison



(b) TCP and SL Comparison 3D



(c) TCP and SL Comparison, loss against the bandwidth.



(d) TCP and SL Comparison, RTT against the bandwidth.

Figure 6.3: SL Adaptation to Loss

region where the user setting is 1500 Kbps, SL uses less bandwidth than TCP. At lower user settings the SL algorithm is less aggressive. So, there are a wide range of RTTs and loss regimes where it is less aggressive than TCP, but at the cost of degraded application performance. The same information for a number of different throttle values is shown in figure 6.3b. The x-axis shows the RTT in seconds, the y-axis shows the probability of loss for each packet and the z-axis shows the throttle value in Kbps. The boundary of the region shows where the Second Life throttle system is producing a TCP fair value.

The comparison between the bandwidth used by Second Life and a TCP fair connection is shown in figure 6.3c. It shows bandwidth against loss rate with a fixed round trip time of 100 ms. The x-axis is the probability of loss and the y-axis is the bandwidth in kbps. It assumes that the loss is constant and correctly detected by Second Life. This shows that Second Life's throttle system is only TCP fair under certain conditions, in this case only when there is very little loss or the loss level is over 3%.

The bandwidth against round trip time for a loss rate of 1% is shown in figure 6.3d. The x-axis is the round trip time in seconds and the y-axis is the bandwidth in kbps. Second Life doesn't take the RTT into account when it calculates the rate, so it will only be TCP fair for small round trip times.

If the sending rate is left at its default value the maximum rate that the server will send packets at will be 750 kbps. The client starts with the value adjusted to its maximum so 750 kbps will also be the starting rate. If the packet loss rate never exceeds 3% then this value will not be adjusted. For an error rate of 3% and a round trip time of 100 ms the TCP fair sending rate will be around 900 kbps. In this case TCP would allow 150 kbits more to be sent every second than Second Life. If the packet loss rate increases to 4% the TCP fair sending rate will decrease to around 760 kbps, however this is over the 3% causing Second Life's system to reduce its sending. If the loss continued at this rate Second Life would reduce its sending rate to 50 kbps, which is significantly lower than the TCP fair sending rate. If the round trip time is increased to 150 ms and packet loss rate is 3%, giving a TCP fair sending rate of around 600 kbps, Second Life be still send at 750 kbps. In this case Second Life would achieve a higher sending rate than the TCP fair value.

As Second Life uses a rate based system which does not take RTT into account, it is more likely to stray outside

the bounds of fair resource utilisation when round trip times are high.

6.1.5 Summary of the comparison of Second Life and TCP adaptation systems

Second Life's throttle adaptation system works quite well for a range of network conditions. However it can lead to situations where Second Life uses more than its fair share of network resource when congestion is moderate, less than its fair share when congestion is high and less than its fair share on a well provisioned network. It does not take the RTT into account when it calculates it rate, so it will not react to changes in network conditions until loss occurs. When there is no loss the Second Life system will not probe for more bandwidth which may be available consequently the user experience may be unnecessarily curtailed.

These problems can be addressed by an adaptive network traffic management system. The challenge is therefore to design an SL client which will allow the network to be probed for bandwidth, which will respond to relatively low levels of loss and which will take a connection's round trip time into account. Such a client would both improve the application performance and responsiveness to other traffic.

6.2 Design of Control Algorithm

There are several possible algorithms that provide congestion control that is TCP fair in that its long term bandwidth utilization is equal to TCP or that its utilization when under large amounts of loss must not exceed TCP. Such algorithms could be rate or window based. This value could use a formula or a reactive system or a combination of the two to determine the limit. Clark and Tennenhouse discuses the design considerations for different network protocols in [32].

6.2.1 Window tracking systems

A window tracking system reacts to events using a function to change the congestion window in response to these events.

There are a class of congestion control algorithms, called Binomial Congestion Control Algorithms[9], that are window tracking and increase and decrease their window based on these equations. This set of congestion control algorithms includes TCP.

$$\begin{aligned} \textbf{I:} & w_{t+R} \leftarrow w_t + \alpha / w_t^k; \alpha > 0 \\ \textbf{D:} & w_{t+\delta t} \leftarrow w_t - \beta w_t^l; 0 < \beta < 1 \end{aligned} \tag{6.2}$$

Where I is the increase function and D is the decrease function, w_t is the window value at time t, R is the RTT and α, β, k and l are constants. A binomial algorithm is TCP-compatible if and only if $k + l = 1$ and $l \leq 1$. For $k = 0$ and $l = 1$ the resulting system is AIMD, like TCP in the congestion avoidance state. For $k = -1$ and $l = 1$ we get Multiplicative Increase Multiplicative Decrease (MIMD) which is what TCP uses in the slow start phase. Chiu and Jain discuss different increase and decrease functions in [27].

The congestion window provided by these algorithms can then be used in one of two ways. It can be used directly to limit the number of packets in flight at anyone time or it can be converted into a rate.

Window based Apart from TCP there are other possible window tracking based controlled congestion control systems.

A system could use the same or similar increase and decrease functions but be packet rather than byte based. For example, Datagram Congestion Control Protocol (DCCP)'s Congestion Control ID (CCID) 2 [57] is designed to approximate the behaviour of the congestion control system used by TCP [160, 4]. A congestion window is maintained to control the number of packets that are in the network at anyone time, as DCCP is packet-oriented rather than byte-oriented, like TCP, the congestion window is in units of packets rather than bytes. TCP does not explicitly provide congestion control of the acknowledgements, though a change in the rate of data packets will also change the number of ACKs. CCID 2 provides for the ratio of ACKs to data packets to be changed when the loss of ACKs is detected. CCID 2 requires that acknowledgements contain an ackvector option that indicates the arrival status of packets.

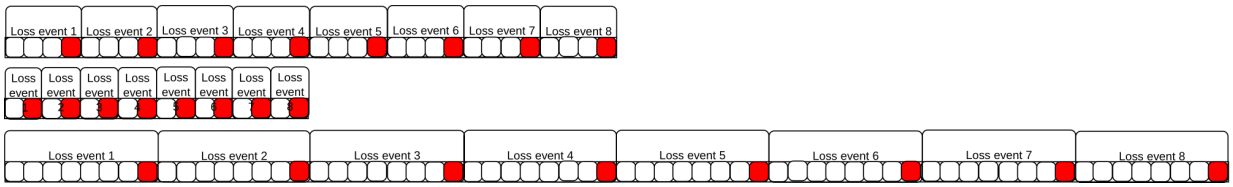


Figure 6.4: Different loss rates.

The window tracking congestion control system provides congestion control that is similar to that provided by TCP. It reacts quickly to changes in the network conditions and is TCP fair. It uses the most bandwidth possible whilst staying TCP fair. It is good for traffic that has large amounts of data to move but does not need a steady transfer rate.

Rate based These systems would work by using the window value maintained to calculate a rate at which to send packets. This can be achieved by limiting the number of packets or bytes that can be sent in a period of time or by waiting a calculated amount of time between sending packets.

6.2.2 Formula controlled systems

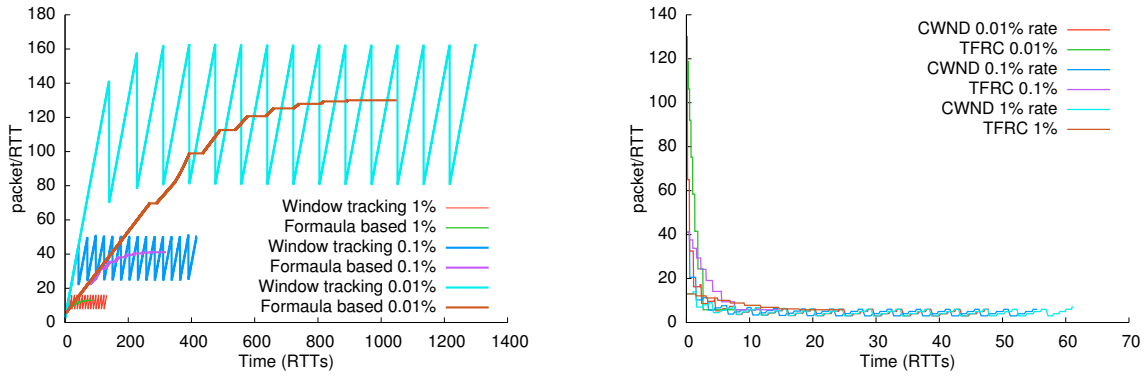
The accuracy of determining the rate or probability of an event dependence on the probability of the event. High probability events can be more accurately and more quickly determined. Using old events means that when the probability changes from high loss to low loss this is detected slowly as old loss events are kept for a long period of time. This results in:

- Loss can be determined more accurately when it is high so algorithms more responsive.
- Loss is difficult to determine when low so algorithms not responsive.
- Loss is over estimated unless it is very high.

An equation based congestion control system based on loss must calculate the probability of loss so that it can determine the correct window size. To prevent oscillation an average is used. In TCP Friendly Rate Control (TFRC) this is a weighted average of 8 congestion events. A loss event starts with a packet loss and then continues for up to one RTT of lost packet. If a packet is lost less than one RTT from the beginning of the loss event it is part of that loss event. The congestion events are recorded as the number of packets from the beginning of the loss event to the beginning of the next loss event, or the current packet for the current unfinished congestion event. This includes the loss event, the lossy part, and the lossless part following it. The congestion events are recorded as a lossy part and a lossless part. The weights for 8 loss events in TRFC are 1.0, 1.0, 1.0, 1.0, 0.8, 0.6, 0.4, 0.2. The values are multiplied by the weights and added together and divided by the total of the weights, this gives the mean distance between losses. The value is calculated for 8 events including the current and for 8 excluding the current one, the larger mean loss interval is used.

Without any averaging it would take a number of packets equal to the loss event rate over 1, the loss event length, to determine the loss rate. With an averaging system using 8 loss events it takes 8 times the number of packets for the value to exactly match the actual value. This means it takes 4 times as many packets to determine that the loss event rate has halved than it does to determine that the rate has doubled. This is illustrated in figure 6.4. The white squares are successfully received packets and the red are lost packets.

If the loss rate changes from one value to another the length of time that this takes depends on the sending rate, so the packet loss before the change. When the change is from a low loss rate to a high loss rate the faster packet transmission rate will cause it to approach the correct value more quickly. The calculated sending rate for a connection when the loss rate has changed from 5% to 0.1% is shown in figure 6.5a. The calculated sending rate when the loss rate has changed from 0.1% to 5% is shown is figure 6.5b. These figures show the difference between a window tracking and equation based congestion control systems in regard to speed of convergence to the correct sending rate. The number of congestion events remembered by the system affects the rate at which the system converges to the correct sending rate. A smaller history will lead to it reaching the correct level more quickly, however a smaller history will also lead to the sending rate being more sensitive to a change in the loss rate.



(a) From 5%

(b) To 5%

Figure 6.5: Calculating the sending rate after a change in loss rate.

Flight size The number of packets in-flight is important as well as the number of packets transmitted per unit time. The congestion window limits the number of in-flight packets so that the number of packets in the network do not overflow the buffers on the routers in the network. The amount of packets on the network divided by the amount of time it takes for those packets to reach there destination and be acknowledged, the RTT, determines the rate at which the packets are delivered and so also the rate at which they are sent onto the network. The AIMD window tracking congestion control of TCP and others converges on a fair share of the number of in flight packets.

The TCP fair equation determines the rate at which a connection will, on average, be sending packets when it has reached a steady state with packets in flight in the network. If a connection is able to transmit packets onto the network a steady rate for sufficient time, without network conditions changing, it will achieve its calculated transmission rate.

If the amount of congestion on the network decreases the RTT will decrease and the in flight packets will be delivered faster. Connections with more packets in-flight than would be necessary to achieve their sending rate without congestion will then be able to more quickly adapt to congestion.

Rate Based An example of a rate based formula systems is DCCP's CCID 3 [58], which is a slight variation of TFRC [72]. It is designed to compete fairly with TCP whilst providing a more smooth packet sending rate. It uses the following formula to determine the correct sending rate:

$$X = \frac{s}{R * \sqrt{2 * b * p/3} + (t_RTO * (3 * \sqrt{3 * b * p/8} * p * (1 + 32 * p^2)))} \tag{6.3}$$

X is the rate at which data is to be transmitted at in bits/second.

s is the packet size in bytes.

R is the round trip time in seconds.

p is the loss event rate, between 0 and 1.0, of the number of loss events as a fraction of the number of packets transmitted.

t_RTO is the retransmission timeout value in seconds that TCP would have calculated.

b is the number of packets acknowledged by a single TCP acknowledgement.

The packet transmission rate is also limited to doubling once for each feedback packet received.

Window based The formula is used to calculate the congestion window value to be used by the congestion control system. It would still require that packets were acknowledged as they arrived otherwise it would not be able to keep the correct number of packets in the network. Such a system would react to changes in the RTT without having to recalculate the value of the congestion window.

6.2.3 Design Choices for traffic management system

In the existing Second Life traffic control system the rate limiter (RL) on the server side of the connection limits the rate at which data is sent from the server to the client. The Second Life rate calculator (SLRC) on the client end of the connection calculates a rate and sends it to the server. The structure of the system is shown in figure 6.6.

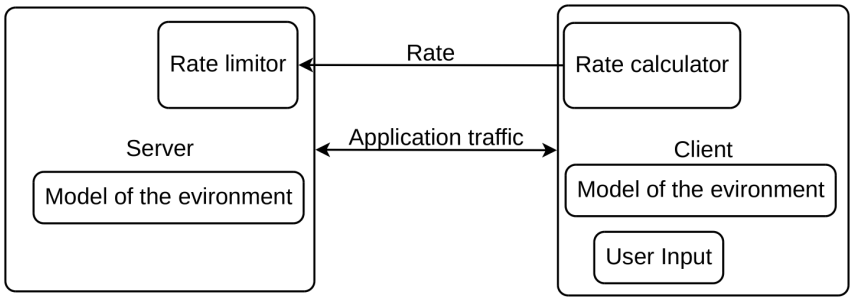


Figure 6.6: The existing system.

The new system involves changing the client but not the server, so it will change the value sent from the client to the server. This limits the type of congestion control system that can be used to those that are rate based rather than window based. There are two types of rate based congestion control that were discussed in sections 6.2.1 and 6.2.2. Equation based congestion control systems adapt to changes in the loss rate more slowly than window tracking systems. Equation based systems will reach a stable level when conditions on the network do not change. Window tracking systems reach a stable equilibrium between an upper and lower bound. An equation based system would then work well for a system that prefers stable rate over the highest mean transfer rate.

The Second Life rate system is in bits per second which means that the congestion control systems will have to use bits and not packets as its unit. All of the types of congestion control system need a packet size in order to give values in bits and not packets. TCP uses the maximum segment size, however TCP is designed so that it will not send out large numbers of packets that are not the maximum segment size. The number of packets as well as the number of bytes is important so a packet size that accurately reflects the size of the packets being sent is important. The average packet size would therefore seem like the correct value to use, however the mix of packet sizes is not constant so a pre-calculated average may not produce correct results.

Second Life circuits also have sequence numbers which can be used to calculate the number of packets lost. The loss detection system used by Second Life is a timeout based system that operates at the receiving end of the connection and relies on detecting gaps in the sequence range. The timeout is 16/1000 times the average ping time in milliseconds, which is 16 times the round trip time. The system keeps track of the sequence number that it expects to see next. If this sequence number is seen then the value is incremented otherwise the sequence number is searched for in the list of sequence numbers of potently lost packets. If it is found this sequence number is removed from the list. If it is not found, and the packet is not marked as having been re-sent, all of the sequence numbers between this one and the expected one are added to the list of possibly lost, and the next expected sequence number is set to this packet sequence number plus one.

The new system, implemented at the client, will be a different rate calculation system called TCP compatible rate calculator (TCRC). TCRC is a rate based window tracking system, maintaining a congestion window and then calculating a sending rate from this window value and RTT. The loss detection system in Second Life does not detect two different type of loss event, just the one. The window tracking system used is modelled after TCP, it maintains a congestion window (CWND) and a slow start threshold (SSTHRESH). The implementation therefore only uses one type of loss event. When loss is detected the CWND and SSTHRESH are reduced to half the CWND. When a packet is received that is not out of sequence the CWND is increased. When the CWND is less than SSTHRESH the CWND is increased by one average packet size. When the CWND is larger than SSTHRESH then CWND is increased by the packet size squared divided by the CWND. This is the commonly used CWND growth function used by TCP [4]. The CWND is supposed to grow by one segment per RTT, this formula approximates that behaviour. It would have been possible to increase window by one segment for each window of packets by counting the number of packets since the CWND was last increased. It is also possible to record the time when CWND was last increased and increase it by one segment every RTT. The method used is a commonly used method and is the simplest method as it requires the least state. The rate is calculated when the system is idle, the rate is calculated by dividing the CWND by the RTT.

The rate at which this information is sent to the server will also be important. The interval at which this calculation is performed is also important. The value can be recalculated every time a packet is received or every time that a packet is received or every time that packet loss is detected. The system could alternatively calculate the sending rate at a set interval or when the loss rate or RTT changes by a large enough value. The system used for this calculation

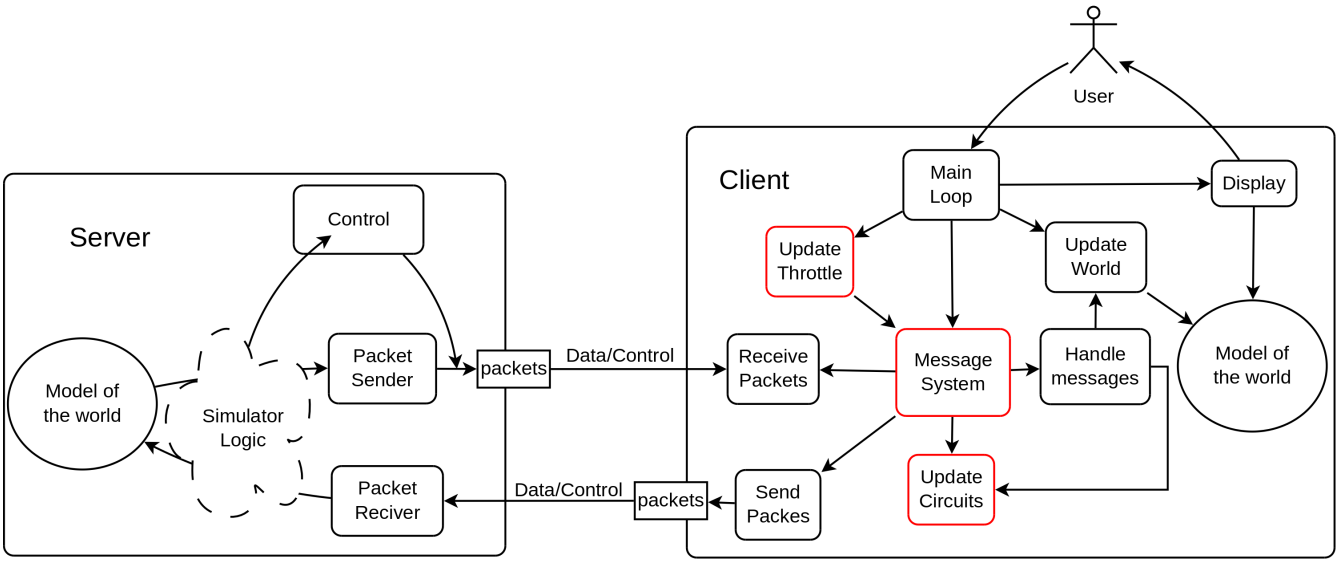


Figure 6.7: The data-flow in the client and server.

will be the window-rate system discussed in the previous section. Such a system would have the advantage of only requiring that the client was changed. The normal design for a congestion control system is to have the system that calculates the rate at which packets or bytes are to be sent so that it can react without the need for the other end of the connection to tell it to.

The periodic calculation method will provide a stable transfer rate whilst adapting to network conditions. The existing system will be replaced with a window tracking system to calculate a throttle total value, this value will then be filtered to prevent sending updates for small changes in the network conditions. The calculated sending rate will be put into the existing function, that calculates the individual channel throttles from overall throttle value. The new system will provide a sending rate that is TCP fair.

6.3 Mongoose structure

This section describes the modified Second Life client, Mongoose.

The Second Life client is implemented in C++ [181], it has several parts that interact to operate. An overview of the functional components is shown in figure 6.7. The red parts are the parts that have been changed. The user looks at the display and uses a mouse, keyboard and possibly a joystick to enter input. The AppViewer component calls the others to perform their operations and holds references to the global objects it needs. The display part performs the rendering of the world and holds the data structures that are specific to display. The Update Throttle part holds the current total throttle value as well as the data structure containing the current calculated throttles for the individual channels. This part has been modified and contains the CWND mean value. The message system is the central part of the message sending and receiving system. It contains the statistics for the network and the references to the packet construction and decoding class and objects. It is also holds the references to the communication objects so that the other parts can access them. This part has been modified to contain the CWND and SSTHRESH values. The update Circuits part handles the circuit data structures, allowing the resending of lost packets and the detection of loss and RTT calculation. It contains the map of circuits, mapping port and IP address to circuit data structures. The circuit data structures contain a map of potentially lost packets, mapping sequence numbers to the time they were added to the map. It also contains the not yet acknowledged reliable packets and the ping time and last time a ping was sent for that circuit. This part has been modified and performs additional loss detection and updates the CWND value in the message system. The receive packet and send packets part interact with the operating system sockets to send or receive packets. The handle messages part decodes packets and calls functions to handle them. It contains the objects representing each type of message. These objects contain a representation of the data structure of the message and a function to call when they have been decoded. The model of the world part contains the objects representing the Second Life objects as well as the ground. The update world contains the functions that update the model of the world in response to the passage of time and messages from the server.

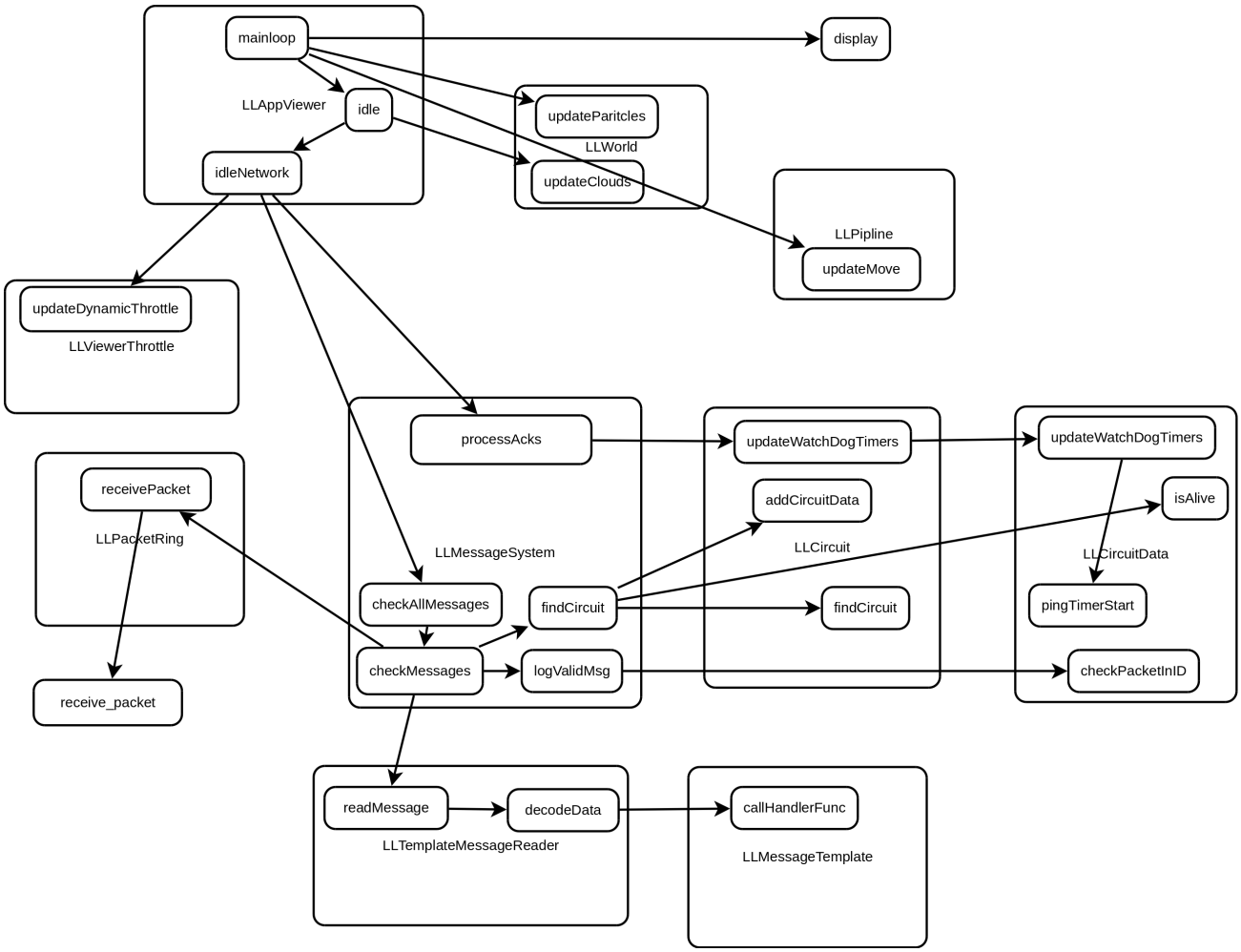


Figure 6.8: A call-graph of the client.

The AppViewer calls the Update World to perform the updates that don't need input from the server, this updates the client model of the world. The AppViewer calls the Message System to pass it user input to send to the server and to get it to receive packets from the server. The Update Throttle part is called to calculate the new throttle value, it gets the information it needs from the message system and then sends the packet to the message system to be sent to the server. The Message System handles the sending and receiving of packets. It assembles packets to be sent out and updates the circuit information, then it passes packets onto the packet sending part. The message system calls the packet receiving part. The packet receiving part reads the packets from the OS socket and puts in the Message System. The Message system updates the circuits with received packets and calls the Handle Message part which calls callback functions from the Update World part to perform the updates that need input from the server. The Handle Messages part also calls callback functions from the update circuits part to update the circuit state. The AppViewer part calls display part to display the model of the world to the user.

The structure of the server is not known in detail. The server has part that sends packets and a part that receives packets. Packet receiving part will feed this information into a component that updates the model of the world and the packet sending part will get information about the world and send it to the client. There is also a rate control part which controls how fast different types of packet are sent to each client.

The `main` function calls the `mainloop` function which calls the other parts of the program. A call graph for the relevant functions is shown in figure 6.8. The `mainloop` calls the display function of draw the environment and `updateParticles` and `updateMove` functions to update it representation of the world. The `mainloop` function also calls the `idle` function which performs the functions that are performed when the client is not rendering or reacting to user input. The `idle` function performs other updates and calls other idle functions including `idleNetwork`, which performs the network functions.

The `idleNetwork` function calls functions to read packets and process them, calculate and send the throttle updates

and time out lost packets. The idleNetwork function calls the checkALLMessages function in the LLMessageSystem class. The checkALLMessages function calls the checkMessages function which checks Second Life messages. The checkMessages function goes into a loop to handle received packets, it first calls receivePacket. The receivePacket function can either create a LLPacketBuffer which calls receive_packet to read the packet from the network, which is then placed on the packet ring and a packet is removed from the ring and returned or it can call receive_packet directly and return this packet. The checkMessages function then dezerocodes the packet is necessary and finds out the sender and calls the findCircuit function. The findCircuit function in LLMessageSystem calls the findCircuit in LLCircuit class to find the LLCircuitData object representing the circuit. If the circuit is found it is checked to determine if it is alive and set to alive if not, if the circuit is not found then a new LLCircuitData object is created for the new circuit. After checkMessages has check that the packet is valid and allowed to be sent, it calls logValidMsg which does some accounting of the number of received packets and calls checkPacketInID of the LLCircuitData object representing the circuit.

The checkPacketInID takes the sequence number of that packet and if it is marked re-sent. The function examines the sequence number to determine if the packet is in-sequence or not. If it is not then it is either filling in a gap or creating one. When the sequence number is larger than expected the sequence numbers in the gap are added to the potentially lost list along with the time. When the sequence number is that of a packet on the potentially lost list it is removed from the list. The checkMessages function then calls the readMessage function which calls the decodeData function which decodes the packet using the LLMessageTemplate object found during the validation process. The decodeData function then calls callHandlerFunc function on the LLMessageTemplate object, which calls the function associated with type of message.

The message handling functions are set using the setHandlerFunc on the LLMessageTemplate function which is called by the setHandlerFuncFast function of the LLMessageSystem. The functions are passed the LLMessageSystem object which they use to get the packet, an example of a handler function is process_layer_data which handles LayerData packets. Another example is the process_complete_ping_check function that processes ping response packets. It stops the ping timer and calculates the ping time.

After the idleNetwork function has finished with processing incoming packets it calls the processAcks function in LLMessageSystem which calls the updateWatchDogTimers function which calls the updateWatchDogTimers function in the LLCircuitData objects for all of the active circuits. This function goes through the list of potentially lost packets to find the ones that were added more than the time out time ago, removes them from the list and increases the number of lost packets. The updateWatchDogTimers in LLCircuitData also sends Ping packets and starts the ping time timer. The processAcks function also calls resendUnackedPackets and sendAcks functions. The idleNetwork function then calls the updateDynamicThrottle function to update the throttle value and send it. The updateDynamicThrottle function checks if sufficient time has elapsed and then increases or decreases the throttle value if the loss level is sufficiently high or low. It then calculates the individual channel throttle values and sends them to the sim where the user's avatar is.

6.3.1 The throttle calculation system

This client includes a window tracking rate calculation system and is based on the Second Life client. The modified functions are described in this section.

When the client receives packets they are parsed to determine the if they belong to any of the existing circuits and if so to which circuit they belong. The packet is then passed to the checkPacketInID which checks the sequence number of the packet against the state of the connection to determine if it is valid and if its arrival could indicate gaps in the sequence space of the circuit. The packet is then passed onto the function that processes its type of packet.

Loss detection

The detection of loss and changing of the value of the CWND happens in a function called checkPacketInID and is called to process the ID of a packet that has been received. The function performs the check to determine if there are gaps in the ID space. When the number of possibly missing packets gets too high this is considered a loss event. If the function determines that the packet is not out of sequence so there is no packet lost the CWND is increased.

The checkPacketInID function takes the sequence number of the packet received and if it has been re-sent. It first checks to see if the packet's id would indicate the sequence number has wrapped around or if the packet is from before the previous wrap around. If it has wrapped around then the maximum id is set to the current value, if it is

from before the maximum is left at its current value and if no wrapper around is involved the maximum is then set to the maximum of the current value and the old maximum. The function then checks if this is the first packet in the circuit if it is the number of packets received is increased by one and the next expected id is set to the packet's ID plus 1 modulo the maximum id, the wrapId is also set to this value.

The function then checks for a gap in the sequence numbers. If the id is the expected next value, indicating no new gap, the value is incremented and wrapped around, the CWND is then compared to SSTHRESH. If CWND is less than SSTHRESH newPackets is increased by one and set to itself modulo 2. If newPackets equals 1 CWND is increased by one packet size. If the CWND is greater than or equal to SSTHRESH CWND is increased by the packet size squared divided by CWND, this approximates increasing CWND by one packet size per RTT. If the packet's id is less than the wrapId this means that first packets in the circuit are out-of-order, this is ignored.

```
//Increase the congestion window by the correct amount.
if(msg->cwnd < msg->ssthreash) {
        msg->newPackets++;
        msg->newPackets %= 2;
        if(msg->newPackets == 0) {
                msg->cwnd += msg->MSS;
        }
} else {
        msg->cwnd += msg->MSS*msg->MSS/msg->cwnd;
}
```

The third case is that there is a gap in the sequence space. The list of possibly lost packets is searched to find out if this is a previously missing packet, if it is it is removed from the potentially lost list and the function returns. The function then checks if the re-sent value is true and ignores re-sent packets. If the id is greater than the expected value the packets in the gap are added to the potentially lost list with the current time. The size of the gap is then added to the global total of sequence gap. The next expected packet is then set to the id plus 1 modulo the maximum id. Outside of the if else statement that detect loss the gap is examined. If it is larger than zero then the number of packets after a gap is increment by one and then compared to NUMPACK, which is 3. If the number of packets after the gap is greater than or equal to NUMPACK then CWND is set to half its previous value and SSTHRESH is set to its new value. The gap packet after gap variable are then set to zero.

```
//Is there a gap in the sequence space.
if(msg->gap > 0) {
        msg->packets_after_gap++;
        //Check the size of the gap against the
        if(msg->packets_after_gap >= NUMPACK) {
                //Reduce the cwnd to half and set the ssthreash to the new
                //cwnd value.
                msg->cwnd = llmax(msg->cwnd/2, 1500);
                msg->ssthreash = msg->cwnd;
                llinfos << "Loss detected window: " << msg->cwnd << llendl;
                //Set the gap and packets after the gap counters back to zero.
                msg->gap = 0;
                msg->packets_after_gap = 0;
        }
}
```

The potentially lost list is examined in the updateWatchDogTimers function. The timeout time is calculated as the minimum of 16 time the RTT and the maximum timeout time. The time now is fetched, then the function loops through the list of potentially lost packets. The time between now and when the packet was added to the list is calculated and compared against the timeout value. If the gap is larger than the timeout the packet is removed from the list and local the global numbers of lost packets are increased by one. This is left in for the statistics that Second Life maintains.

Throttle calculation

The calculation of the rate happens in the function updateDynamicThrottle. The existing throttle calculation code has been removed and replaced with code that calculates rate using this equation 6.4. The value is clamped between the upper and lower limit and then checked against the previous value, if it is the same then no new value is sent.

When the client is not processing any events it runs through the idle loop functions these perform periodical tasks. The updateDynamicThrottle function is part of the idle loop it performs the throttle value calculation.

$$X = \frac{CWND}{RTT} \tag{6.4}$$

The updateDynamicThrottle function adds the CWND value to a total and increments the number of CWND samples. The function then checks the update time to determine if sufficient time has elapsed since the last time the throttle value was adjusted, if it has not the function returns. The function then resets the timer and calculates the mean CWND value from the CWND total and number of samples and sets the CWND total and number of samples to zero. A copy of the old rate value is recorded as old_bandwidth. The CWND value is calculated by dividing CWND by the RTT and the new throttle value is sent to the server. The following is the contents of the updateDynamicThrottle function:

```
int cwndMean;

LLMessageSystem* msg = gMessageSystem;
//Save the old bandwdith value.
msg->cwndTotal += msg->cwnd;
msg->cwnd_samples++;

if (mUpdateTimer.getElapsedTimeF32() < DYNAMIC_UPDATE_DURATION)
{
        return;
}
mUpdateTimer.reset();

cwndMean = msg->cwndTotal / msg->cwnd_samples;
msg->cwndTotal = 0;
msg->cwnd_samples = 0;

F32 old_bandwdith = mCurrentBandwidth;
F32 new_bandwdith = mCurrentBandwidth;

//Get the mean ping time.
F32 ping_time = LLViewerStats::getInstance()->mSimPingStat.getMean();
//Calculate the new bandwidth value
new_bandwdith = llmax((cwndMean*8000.f)/ping_time, 1500.f);
new_bandwdith = llmin(new_bandwdith, 1536000.f);
//Has the bandwidth allocation change since last time it was calculated
int old_b = ((int) old_bandwdith) / 51200;
int new_b = ((int) new_bandwdith) / 51200;
if(old_b != new_b)
{
        mCurrentBandwidth = new_bandwdith;
        //Calculate the amount of bandwdith the each of the throttles is to be
        //allocated from the total bandwidth.
        mCurrent = getThrottleGroup(mCurrentBandwidth/1024.0f);
        mCurrent.sendToSim();
        llinfos << "Setting network throttle to " << mCurrentBandwidth << llendl;
        llinfos << "window: " << cwndMean  << llendl;
```

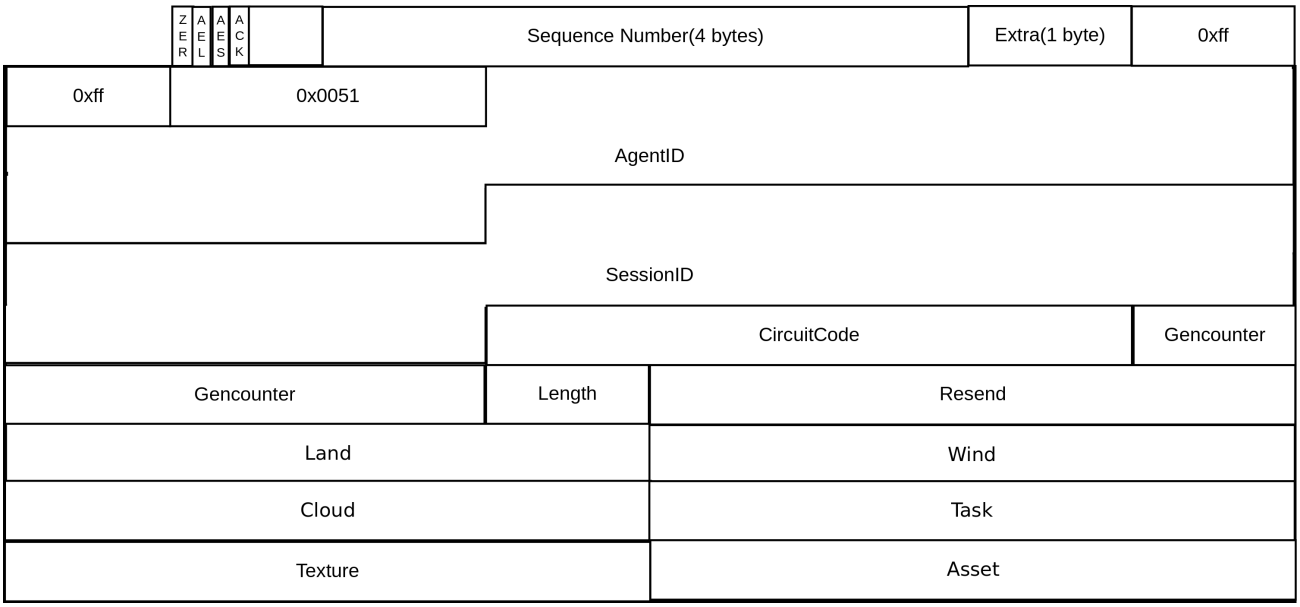


Figure 6.9: The structure of an AgentThrottle packet.

```
        llinfos << "RTT: " << ping_time << llendl;
}
```

Case studies

ObjectUpdate When an ObjectUpdate packet is read by `receive_packet` it passes it back in the supplied buffer to `receivePacket`. The `receivePacket` packet uses the buffer it was supplied by `checkMessages`. The `checkMessages` function checks if the packet is large enough to be a valid packet. It then gets the sender information and passes this to `findCircuit` to find the circuit. If the circuit has been found and the packet contains acknowledgements the function loops through the ACKs and marks the packets that they acknowledge as acknowledged and remove the circuit from the list of circuit with outstanding reliable packets. The function then reads the reliable and re-sent flags, if it is re-sent it is checked to determine if it is a duplicate. The packet is then passed to `LLTemplateMessageReader`'s `validateMessage` function which passes it to `decodeTemplate` which decodes the packet number fetches the `LLMessageTemplate` object for the packet type, in this case ObjectUpdate, from the map of packet numbers to templates and then sets the supplied ptr to the template. It then returns true to `checkMessages` indicating that the packet is valid. It then checks if packets of this type are allowed and then calls `logValidMsg`, if there is a gap in the sequence numbers loss may be detected at this stage. The packet is then passed to `readMessage` which passes it on to `decodeData`. The `decodeData` function first skips over the packet header it then creates a `LLMsgData` object for the packet. It loops through the blocks that the packet template has and copies the correct data into the correct fields. The function then checks that the decoded packet has any parts and then it has calls `callHandlerFunc` which in this case will call `process_object_update`. `process_object_update` logs the number of bits and calls `processObjectUpdate`.

The `processObjectUpdate` will read the number of object parts that are in the packet, for each it will determine that it is a full update and read its local and global ID from the packet. It then tries to find the object, if it exists it will move it to its current region if necessary. If the object is not found it is created. Assuming the object is not on the dead list the `processUpdateCore` function is called, which first calls `processUpdateMessage` on the object. The `processUpdateMessage` function extracts all of the information from the message and updates the object. If the object has just been created `processUpdateCore` adds it to the rendering pipeline.

Throttle update send The `updateDynamicThrottle` function performs the throttle calculation of the new throttle as described in section 6.3.1. Once the new value is calculated and the difference is determined to be sufficiently large the `getThrottleGroup` function is called with the calculated throttle total value. The `getThrottleGroup` function takes the throttle value and determines which of the sets of throttle values are larger than the current value. If all are, then the lowest set of values are used. If none are then the difference between the calculated value and the total

for the largest set is calculated and the difference between the throttle values for the two largest sets is calculated. The difference between the calculated total and largest throttle values is divided by the difference between the sets of throttle values and then difference between the throttle values is multiplied by this value and the result is added to the largest set of throttle values. If the values is between the throttle values the fraction of the difference is calculated and multiplied by the difference in the throttle values and added to the lower set.

The `updateDynamicThrottle` function then calls `sendToSim` on the returned throttle values. The `sendToSim` function makes calls to the `LLMessageSystem` object to construct a AgentUpdate message. It first calls `newMesageFast` which selects the correct message builder and then calls its `newMessage` function to find the correct message template and creates the message data data structure. The `sendToSim` function then creates the AgentData block and adds the agent and session IDs from the `LLAgent` object and the circuit code from the message system. It then adds the Throttle block and adds a GenCounter value of zero. It packs the throttle values into a block of memory and then adds it to the packet data structure. Figure 6.9 shows the structure of the packet. The function then calls `sendReliableMessage` on the `LLAgent` object. This function checks that the client is connected and that Avatar is on a region and then calls `sendReliable` in the message system, passing it the host of the region were Avatar is located. The `sendReliable` function calculates the timeout for resending the message and then calls `sendMessage`.

The `sendMessage` function first `buildMessage` on the message builder, which serializes the message. The function then finds the circuit for the host and would create one if it didn't exist. It then gets the next packet ID from the circuit and put it in the packet. The packet is now compressed so the function calls `compressMessage` which will zero-code the packet. This message is reliable so the function now sets the reliable flag and adds the packet to the circuit list of reliable packets awaiting an ACK. The function then checks is there is space on the packet for ACKs and if there are ACKs to be sent. If there are ACKs to send and space on the packet then the ACK flag is set and as many ACKs as will fit and exist are appends. It then sends the packet.

RTT calculation The `processAcks` function in the `LLMessageSystem` part calls the `updateWatchDogTimers` function, checks the timers a sorted list of the `LLCircuitData` objects and if their ping send timers have run out calls the `updateWatchDogTimers` function on them. This function goes through the list of potentially lost packets and also sends Ping packets and starts the ping time timer. It first calls `pingTimerStart`, which sets `mPingTime` to the time from the message system and then increments the number of in transit pings. It also checks if there are too many in transit ping and if so sets the connection to blocked. The `updateWatchDogTimers` function then constructs the StartPingCheck packet. It calls the message systems `newMesageFast` to create the packet and then adds the PingID block. This block holds the ping ID and ID of the oldest unacked packet. The function then calls `sendMessage` which builds the packet, updates the circuit, adds ACKs if there are any and sends the packet. The server will receive the StartPingCheck packet and reply with a CompletePingCheck packet. The CompletePingCheck packet is read in by the `receive_packet`, its circuit is then found. The packet is then validated, the circuit is updated and the packet is decoded. The `callHandlerFunc` calls the `process_complete_ping_check` function to process the message.

The `process_complete_ping_check` function reads the ping ID from the message, then finds the circuit that the packet belongs to. If the circuit is found the `pingTimerStop` function is called on its object. The `pingTimerStop` function calculates the time between the `mPingTime` and the current time from the message system and then sets `mLastPingReceivedTime` to the current time. The difference between the last ping ID and the ID of this reply is calculated. The time for this ping is then calculated as the time between this reply and the last being sent plus the gap in ping IDs time 5 seconds. It then calls `SetPingDelay` passing it the calculated ping time. The `SetPingDelay` function sets `mPingDelay` to the ping time and `mPingDelayAveraged` to the maximum of the ping time and value returned by `getPingDelayAveraged`. `getPingDelayAveraged` returns the minimum of the maximum of the value returned by `getPingInTransitTime` and the value of `mPingDelayAveraged` and the Averaged Ping Maximum value, which is 2 seconds. The `getPingInTransitTime` function set a variable to zero then, if there are pings in transit sets it to the number of pings in transit times the ping interval minus one plus the current time minus the `mPingTime` value. It then returns the variable. The `SetPingDelay` function then sets the `mPingDelayAveraged` to one minus the `LL_AVERAGED_PING_ALPHA`, which is 0.2, times the `mPingDelayAveraged` plus the `LL_AVERAGED_PING_ALPHA` times the ping time. Finally `SetPingDelay` clamps the `mPingDelayAveraged` between the maximum and minimum ping averaged values. The `pingTimerStop` function sets the pings in transit to the difference between the ping IDs and unblocks the circuit if it now has sufficiently few in transit pings.

The averaged ping value is used for calculating the time-out of packets on that circuit. The time used for congestion control is the mean of the ping times from the user's avatar's sim.

6.3.2 Summary of Mongoose structure

The new system uses the same RTT calculations as the Second Life client. The loss detection system is, in the new system, not time out based, but instead works by counting the number of packets after a gap in the sequence numbers. The gap detection system is the same as the Second Life client, with additional code to count the number of packets after a gap and adjust the CWND accordingly the CWND increase is also added to the `checkPacketInID` function. The `updateDynamicThrottle` function is replaced with the TCP fair rate calculation using the RTT value calculated using ping packets.

6.4 Summary

There are a number of different types of congestion control system that can be used by a network application. Not all of these are applicable to all types of application. Window tracking systems provide a larger total amount of bandwidth to the application, so are better suited to applications which have to move information as quickly as possible. Second Life is such an application, as it has texture data to move from the server to the client without requirements of time it arrives.

The window tracking rate based congestion control system for Second Life is able to control its traffic. This system produces a rate which is comparable to the rate an equation rate-based congestion control system produces and is able to use this rate to control the amount of bandwidth used.

Chapter 7

Evaluation

In this chapter the evaluation of the new Second Life client, called Mongoose, is described. The main difference between Mongoose and the standard Second Life client is the mechanism that calculates the throttle value to be sent to the server. The throttle values sent out from the client to the server are presented along with the amount of bandwidth used by the server.

The calculation of the throttle values in response to the network conditions is evaluated. The window tracking throttle calculation system is compared with the Transmission Control Protocol (TCP) fair rate values and TCP Friendly Rate Control (TFRC) [72]. The responsiveness of the server to the throttle value sent from the client is evaluated. The importance of the time interval between calculations of the throttle value was investigated. This chapter is organised into the following sections:

Section 7.1: The methodology employed in this evaluation of Mongoose is described.

Section 7.2: The throttle values calculated by the Second Life client and the Mongoose client are compared and evaluated.

Section 7.4: The amount of bandwidth used is compared to the calculated throttle values. The importance of the interval between rate calculations is examined.

Section 7.5: The summary statistics for the evaluation are presented.

7.1 Methodology

The aim of these experiments is to determine the fairness of the Mongoose client under different network conditions and to compare different congestion control systems.

The methodology was to conduct a series of controlled experiments creating real connections across the Internet to production Second Life servers, where the avatar performs a series of defined actions. The island used was Miramare, which had an average of 12 avatars present and 8749 objects. The traffic that was received and sent from the Second Life client was captured for post processing. The activity undertaken was walking along a set path. During the experiments the network conditions were measured to provide context for the work. Each scenario lasts for ten minutes.

The experiments were carried out from the University of St. Andrews School of Computer Science. Traffic statistics were generated for the global traffic and for individual traffic categories of interest. The results are presented tabularly and through time and distribution graphs. The client's cache was emptied between each run. The time series graphs in this chapter go from 100 second to 200 seconds to allow the details of the stable state to be seen in more detail.

Connections are fed through a router, which was equipped with the netem network emulator [73]. This facilitates control over available bandwidth, levels and pattern of loss, delay and jitter. It is described in more detail in section 3.3.

The modified client was tested under different network conditions. The netem system was used to introduce extra delay and loss into the network. The network used allowed loss to be introduced in both directions. Loss levels of 0.1%, 1%, 3% and 5%, were used and additional delays of 50 ms and 100 ms were introduced. The traffic was captured at both sides of the traffic shaping network, this allows the introduced loss rate per unit time to be calculated accurately.

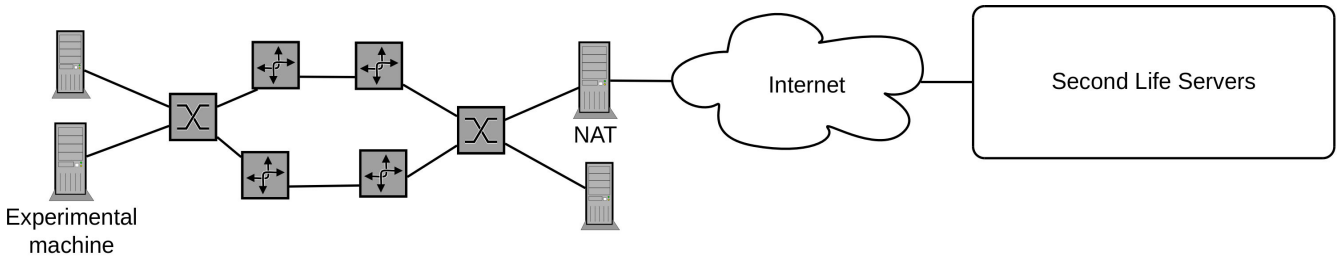


Figure 7.1: The structure of the test network.

The resulting round trip time (RTT) can be calculated from the captured traffic. This allows a TCP fair rate to be calculated and to be compared against the rate value sent from the client and against the rate at which the server sends packets to the client.

The analysis of the Second Life client shows that its throttle system does not react to changes in the round trip time. The different rate based congestion control systems examined include the RTT by dividing the per RTT rate by the RTT. They are affected by the RTT but don't react to changes in it directly.

TCPDump was used to capture the packets. The SLparse program (created specifically for analysing Second Life traffic) was used in the post processing, this program is described in section 3.2.

The traffic for Second Life goes from the client machine to a router to a second router, which sends it on to the network address translation (NAT) box and out onto the Internet. The traffic coming from the Second Life servers come in through the NAT and goes through the other two routers to the client machine. The netem setup runs on two of the routers, one in each direction this allows the traffic to be manipulated in both directions. The queue discipline that are used can only affect out going traffic, unless the traffic is routed through a virtual device on the machine which would affect the traffic.

The traffic is captured at both sides of the test network. The packets lost by the test network are therefore captured before they are dropped. This allows the drops to be detected and the loss rate to be calculated. A TCP fair rate value is calculated by taking the loss rate calculated once every 5 seconds. The loss rate value is then run through the TCP rate equation 7.1 to produce a rate for that level of loss. This produces a TCP fair value for the trace which changes with time, but doesn't change so fast that it doesn't relate to the long term loss rate.

$$X = \frac{1.3 * S}{RTT\sqrt{p}} \tag{7.1}$$

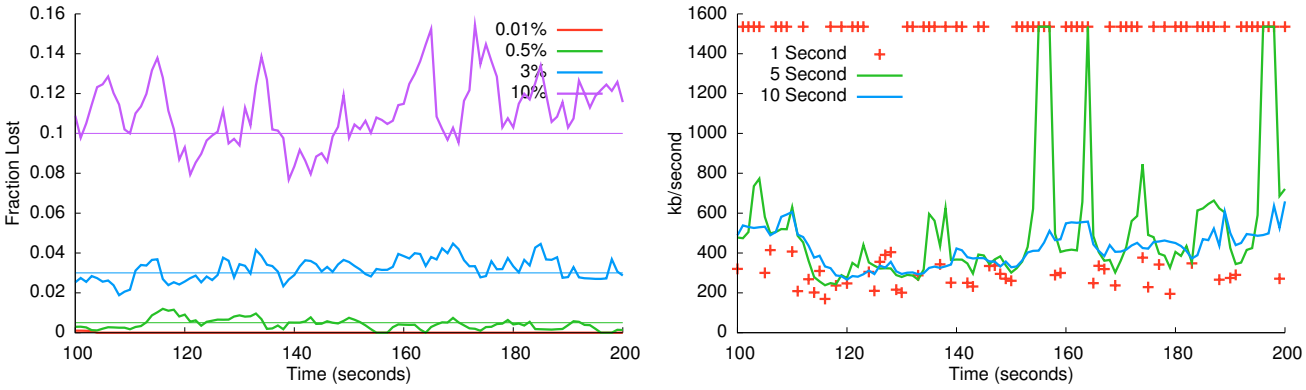
The TFRC [72] value is calculated from a loss level estimation value calculated by running through the trace and calculating the average of 8 loss intervals. A loss interval is the number of packets from the beginning of a loss event to the beginning of the next. This system adapts to increases in the loss rate faster than decreases in it. This loss rate is then put into the TCP fair equation [118], shown in 7.1, to producing a rate.

Spearman's rank correlation coefficient is a non-parametric measure of statistical dependence between variables. It produces a value between 1 and -1. The sign of the value indicates if the association between the variables is positive or negative. Spearman correlation can deal with non linear correlations and data sets with outliers. It is a rank correlation so it is the correlation between ranked variables.

The median loss rate is calculated from the per second loss rate values, as are the percentiles and maximum and minimum values.

The loss rate for different traces recorded with this setup are shown in figure 7.2a. The x-axis is the time in seconds. The y-axis is the fraction of packets lost. The thick lines are the loss rate average over 5 seconds. The thin lines are the set loss rates. The loss rate move around the set loss value but changes significantly over time. This variation in the loss rate is the reason that the measured loss rate rather than the set loss rate are used.

The TCP fair rate values calculated from the loss values for the 0.5% loss rate calculated for different averaging intervals, is show in figure 7.2b. The x-axis shows the time in seconds and the y-axis shows the throttle TCP fair value in kbps. The red line shows the values calculated from loss rate calculated each second, the green line shows the values averaged over 5 seconds and the blue line shows the value for 10 seconds. The loss rate varies significantly second to second because there aren't enough packets per second to have a stable rate. The averages are calculated for each second for all of the averaging intervals using a sliding average. The 5 second version is used because it provides a balance between moving correctness and stability.



(a) Loss levels for different set loss levels.

(b) Different averaging intervals for one session.

Figure 7.2: Loss levels against time.

In summary real connections are created to Second Life and set actions will be performed in world. The traffic is captured and the network conditions are changed. Traffic statistics are recorded and used to evaluate the Mongoose client. Three sets of experiments will be performed. The calculated throttle values will be compared to TCP fair. A comparison of the window tracking system to a formula based system. The TCP fairness of the bandwidth utilization of Second Life.

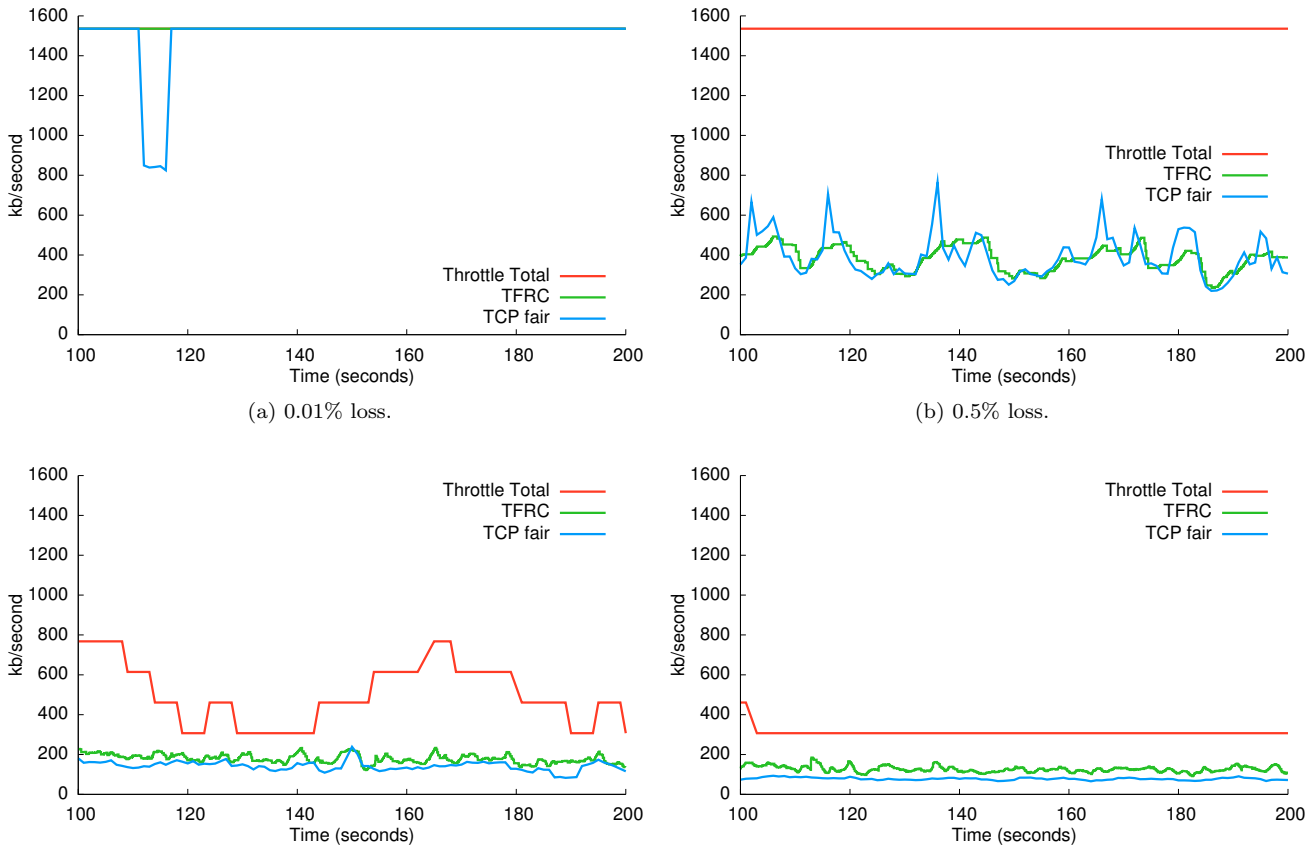
7.2 Calculated throttle values

In this section the ability of the Second Life and Mongoose clients to calculate throttle values are compared. The Second Life throttle values are compared to TCP fair values and TFRC values and presented in graph form. The Mongoose throttle values are then compared to TCP fair values and TFRC values and the compared to the Second Life values. Statistics for the Second Life and Mongoose throttle values are presented in table form. Summary values are then presented in a graph. This allows the user to determine that the client is able to produce a throttle value capable of causing the server to use an amount of traffic that is TCP fair.

Second Life First of all the degree to which the throttle values calculated by the client Second Life are TCP fair is examined. The properties of Second Life traffic have been described previously. The throttle values are compared to what is expected from the static analysis of its throttle system, in section 6.1.

Time series traces for the Second Life client are shown in figure 7.3. The x-axis gives the time in seconds since the beginning of the trace. The y-axis shows the rate in kbps. The red line shows the throttle value extracted from the trace. The green line shows the TFRC value calculated from the trace. The blue line is the TCP fair rate. The behaviour of the SLC throttle adoption system in these graphs represent behaviour in different regions of the graphs in figure 6.3a. The regions of the graph are TCP fair with a loss rate below 3%, TCP unfair with a loss rate below 3%, TCP fair with a loss rate above 3% and TCP unfair with a loss rate above 3%. The user set value for the throttle system in these traces was set to 1,500 kbps. If the value had been set lower the maximum value that it could reach would have been lower, not allow the client to use as much bandwidth when it is available. The traces have around 100 packets per second so for connections with a loss rate of less than 0.5% there will be loss in less than half of the seconds of the connection.

(a) A trace with a loss rate of 0.01% is shown in figure 7.3a. The throttle does not change from the maximum value during this trace as the loss rate detected by the client does not exceed 3%. The throttle values line is below the TFRC and TCP fair lines. The mean loss rate is 0.01%, the maximum loss rate during this trace is 2.6% and the median loss rate is 0%. This is in the TCP fair region of graph with a loss rate below the threshold of 3%. If the throttle value had been set to the default value, of 500 kbps, the throttle would not have reached the TCP fair values. The median throttle value is 1,500 kbps and the average TCP fair value is 1,500 kbps giving a ratio of 1.



(a) 0.01% loss.

(b) 0.5% loss.

(c) 3% loss.

(d) 10% loss.

Figure 7.3: Second Life calculated throttle values and TFRC values at different loss levels.

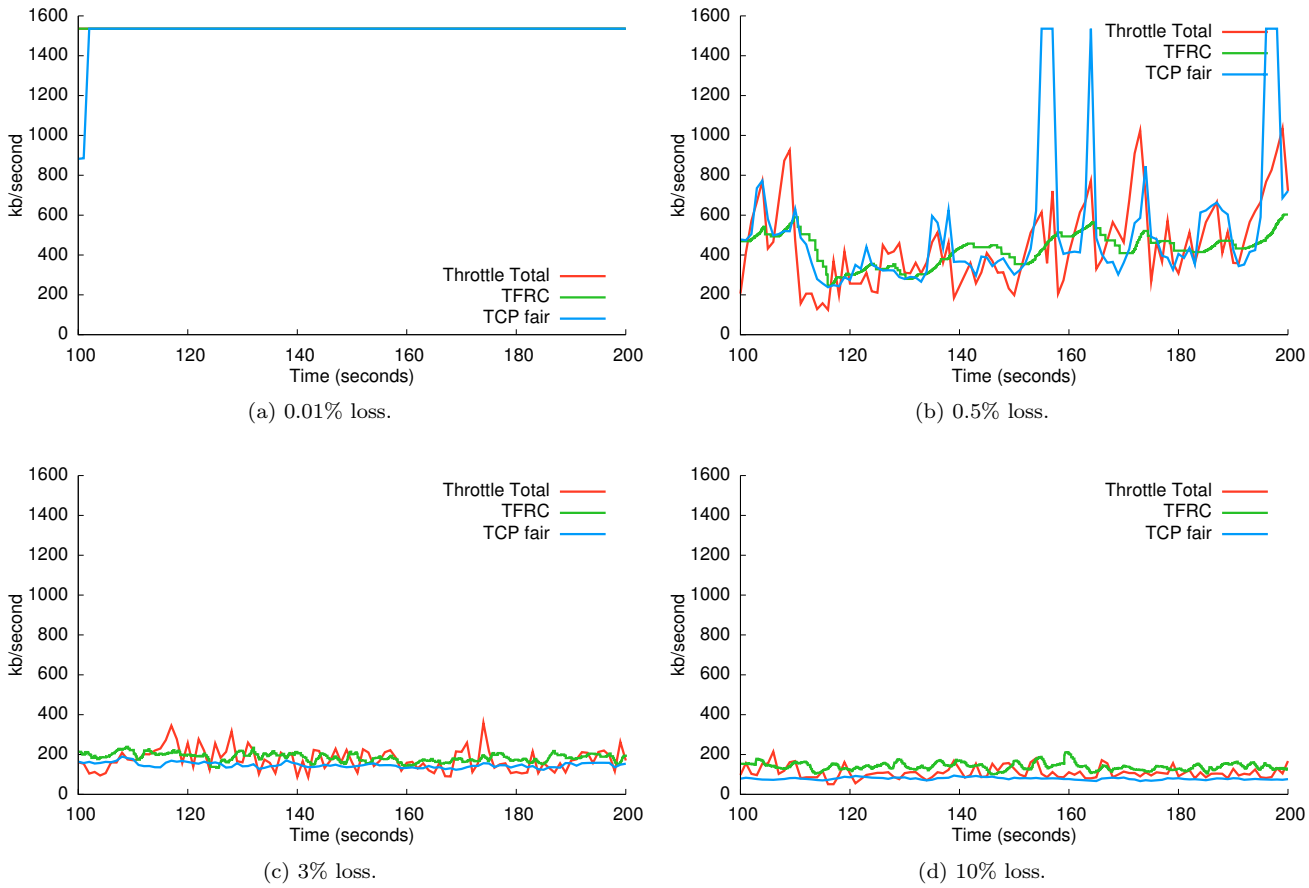
(b) The throttle values for a trace with a 0.5% loss rate is shown in figure 7.3b. The TCP fair values and TFRC values are close together. Like the 0.05% values the throttle value never changes. The mean loss rate is 0.49%, the median loss rate is 0% and the maximum loss rate is 5.8%. This is in the TCP unfair region with a loss rate below 3%. If the throttle value were set lower the calculated throttle value would be closer to the TCP fair value than the values in this trace. The median throttle value is 1,500 kbps and the average TCP fair value is 362.22 kbps giving a ratio of 4.14.

(c) The throttle values for a loss rate of 3% is shows in figure 7.3c. The TCP fair values are closer to the throttle value than for the 0.5% loss rate. The mean loss rate for this trace is 3.43%, the median loss rate is 2.99%, the 25 percentile is 1.67% and the 75 percentile 4.59%. This is in the TCP unfair region with a loss rate above the 3% threshold. The median throttle value is 450 kbps and the average TCP fair value is 140.70 kbps giving a ratio of 3.20.

(d) The throttle values for a loss rate of 10% are shown in figure 7.3d. The throttle value stays at the same level. The mean loss rate is 11.03%, the median loss rate is 10.81%, the 25 percentile is 8.33% and the 75 percentile is 13.64%. This is also in the TCP unfair region above the threshold. The median throttle value is 300 kbps and the average TCP fair value is 76.70 kbps giving a ratio of 3.91.

The Second Life client is not calculating a rate that is close to the TCP fair rate. When the loss rate is below its adaptation threshold Second Life does not change its bandwidth usage. The calculated TFRC values are close to the TCP fair values. When the loss rate is large enough that the TCP fair value is less than the maximum throttle value, as in figure 7.3b, the difference between the throttle value and the TCP values is large. When the loss is over the adaptation threshold the throttle value is larger than the TCP fair values by less than when it is below the threshold.

This confirms the static analysis, at low loss levels the client generates a rate that is below the TCP fair value. From static analysis as the loss level increase the TCP fair rate reduces to below the throttle value, the gap increases



(a) 0.01% loss.

(b) 0.5% loss.

(c) 3% loss.

(d) 10% loss.

Figure 7.4: Mongoose calculated throttle values and TFRC values at different loss levels.

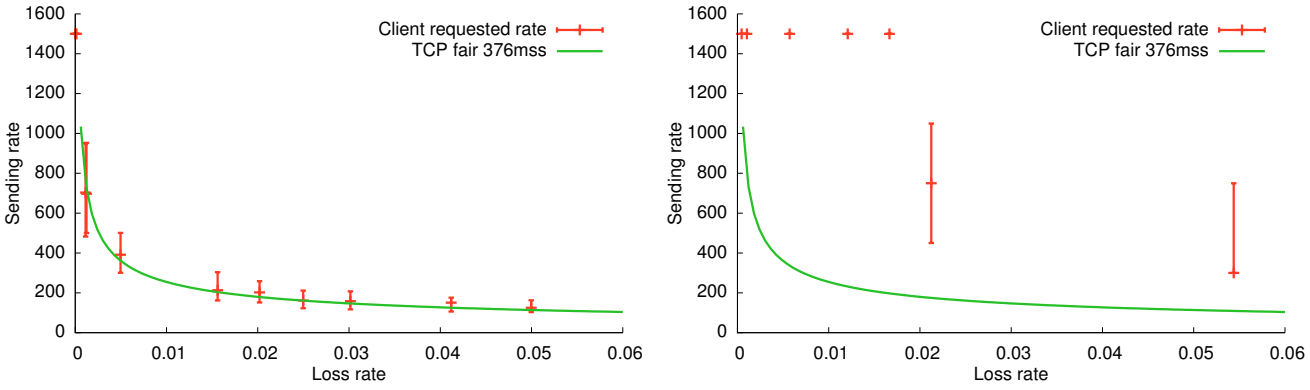
until the loss level reaches 3%. The throttle then goes below the TCP fair value, then as the loss level increases the throttle value goes above the TCP fair value and the gap increases as the loss level increases.

Mongoose Mongoose uses a window tracking algorithm to calculate its requested throttle value. This algorithm mimics the TCP steady state behaviour, the window value is then converted into a rate using the RTT value calculated by the client. The ability of Mongoose to calculate throttle values has been evaluated by running it under different network conditions. These values are then compared against the TCP fair and TFRC values from the traces.

The Mongoose client was run through different network conditions, figure 7.4 shows the time series graphs for four different levels of loss. In each graph the x-axis gives the time in seconds and the y-axis gives the bandwidth in kbps. The red line is the transmitted throttle values extracted from the trace, sampled once a second. The green line is a TCP fair rate control (TFRC) value calculated from the loss information extracted from the traffic. The blue line shows a TCP fair rate calculated from the loss rate calculated every 5 seconds. Both the throttle value and the TFRC values are limited to 1,500 Kbps.

	Second Life				**Mongoose**			
Set Loss rate	0.01%	0.5%	5%	10%	0.01%	0.5%	3%	10%
Measured Loss Rate	0.015%	0.494%	5.466%	11.016%	0.011%	0.496%	3.107%	10.083%
Throttle value (Kbps)	1,500	1,500	450	300	1,500	391.42	156.67	104.96
Average TFRC (Kbps)	1,500	387.20	183.72	126.10	1,500	397.34	182.76	130.28
TCP fair rate (Kbps)	1,500	362.22	140.70	76.70	1,500	361.48	144.43	80.18
Correlation	0.01325	0.01311	0.04931	0.04977	-0.14108	-0.51333	-0.40719	-0.23768
Difference (Kbps)	0	1,137.78	309.3	223.3	0	29.94	12.24	24.78

Table 7.1: Throttle calculation statistics



(a) Mongoose client

(b) Second Life client

Figure 7.5: Calculated throttle values against loss rate.

(a) The throttle and TCP rate values for a loss rate of 0.01% are shown in figure 7.4a. The mean loss rate is 0.01%, the median loss rate is 0% and the maximum loss rate is 1.4%. The TFRC value does not change but the throttle value does, this is because the TFRC value calculated for this connection is more than the limit, so the calculated value decreases but not below the upper limit. This is similarly close to the TCP fair values as in figure 7.3a. The throttle values, TFRC and TCP fair values lines are one top of each other, with the TCP fair line drawn on top. The median throttle value is 1,500 kbps as is the TCP fair value giving a ratio of 1.

(b) The throttle and TCP rate values for a loss rate of 0.5% are shown in figure 7.4b. The mean loss rate is 0.5%, the median loss rate is 0% and the maximum is 5%. The throttle value changes a lot during the trace and is significantly closer to the TCP fair values than in figure 7.3b. This is because the window tracking congestion control system used by Mongoose reacts to any congestion and doesn't have a threshold value. The median throttle value is 391.42 kbps and the average TCP fair value is 361.48 kbps giving a ratio of 1.08.

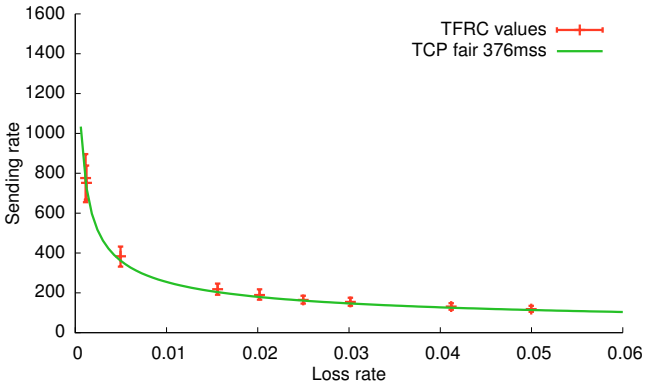
(c) The throttle and TCP rate values for a loss rate of 3% are shown in figure 7.4c. The mean loss rate is 3.23%, the median loss rate is 2.9%, 25 percentile is 1.8%, the 75 percentile is 4.3% and the maximum is 33.3%. The throttle values are closer to the TCP values than in figure 7.3c. The throttle values in this trace are lower than the lower bound that the Second Life system would reach with the settings that were used. The median throttle value is 156.67 kbps and the average TCP fair value is 144.43 kbps giving a ratio of 1.08.

(d) The throttle and TCP rate values for a loss rate of 10% are shown in figure 7.4d. The mean loss rate is 11.25%, the median loss rate is 10.8%, 25 percentile is 8%, the 75 percentile is 13.7% and the maximum is 50%. The throttle values are again closer to the TCP values than in figure 7.3d. The median throttle value is 104.96 kbps and the average TCP fair value is 80.18 kbps giving a ratio of 1.31. The ratio is despite the fact that the values are close together, the absolute difference is small but the ratio is larger as the value are small. The throttle value will not go below its minimum value this can increase the mean value.

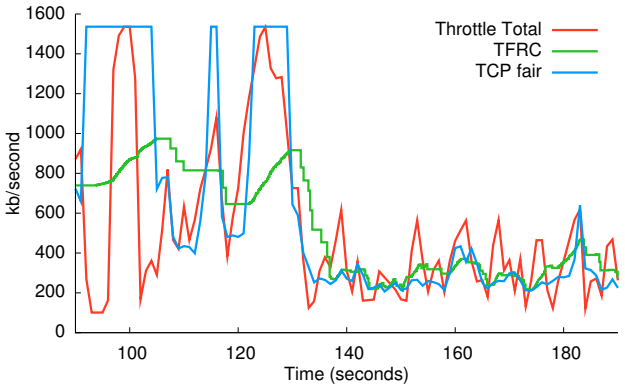
The Mongoose throttle values are closer to the TCP fair values than the Second Life throttle values. This client has no interval between its calculations of the throttle value so it will change more quickly than the Second Life client. The Mongoose is moving above and below the TFRC value and the averaged TCP fair values as it is a window tracking system, so by design it will change quickly and move around the fair value.

Summary Statistics Statistics for the traces in the earlier graphs are shown in table 7.1. The measured loss rate values are the mean loss rates for the traces. The measured loss level for 0.5% loss is 0.494% and 0.496%, the 10% loss rate are 11.016% and 10.083%.

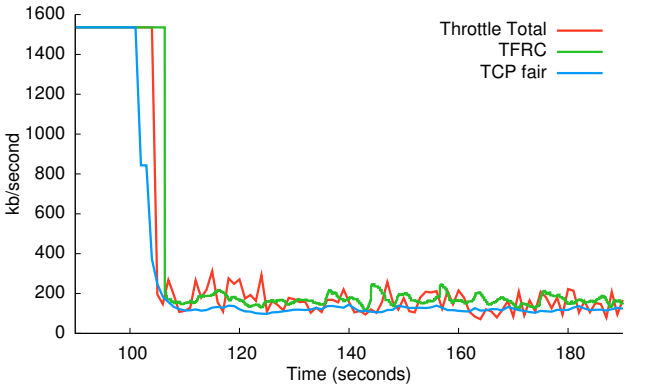
The throttle values are the median value from the trace. The throttle values are sampled once per second from the trace and a median value is than calculated from the traffic. The median value is used because the Second Life client always starts at the maximum value, so the mean value would be more affected by the length of the trace than the median value. The median throttle value for 0.5% loss are 1,500 kbps for Second Life and 391.42 kbps for Mongoose, this is a ratio of 3.93.



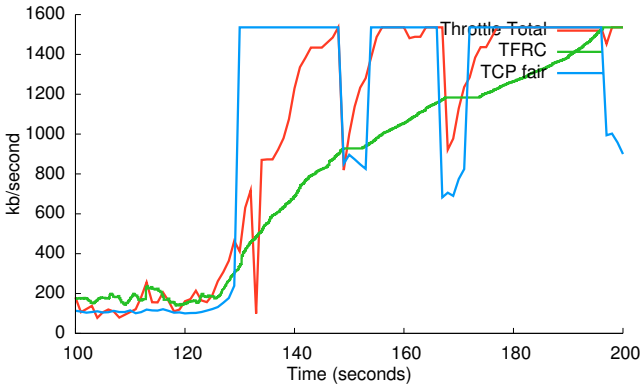
(a) Summary Graph



(b) From 0.1% loss to 1% loss.



(c) From 0.01% loss to 5% loss.



(d) From 5% loss to 0.01% loss.

Figure 7.6: Mongoose calculated throttle values and TFRC values with a change in loss rate.

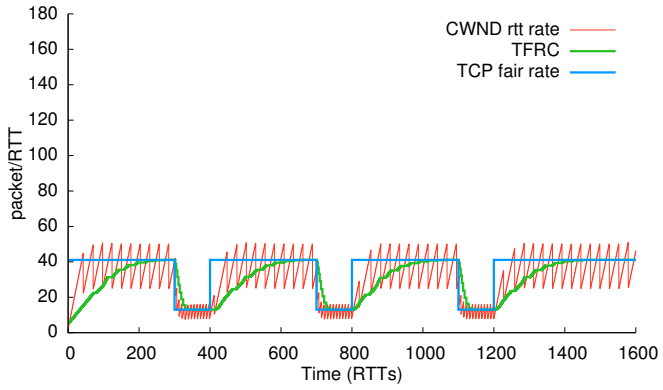
The average TFRC value is the median of the TFRC values calculated for the throttles. The median values were used to match the throttle value. The average TFRC values for 0.5% loss are 387.20 kbps and 397.34 kbps which are close together as expected because the loss levels are close together.

The correlation value is the correlation between the throttle value and the loss rate. The correlation used is Spearman's rank correlation coefficient which is a non-parametric measure of statistical dependence between two variables. This allows non linear correlation to be measured between variables, which the relationship between the loss rate and the throttle values are. The correlation value for the Second Life client is very small and positive. The positive correlation would indicate that a larger loss rate is producing a larger throttle value which is the opposite of the relationship that these two values should have. The correlation values for Mongoose are larger and negative. This is the correct direction for the correlation, meaning more loss means lower throttle values. The Second Life client's correlation for 0.5% loss is 0.01311, it has a probd value of 0.7346 and a probrs value of -19.2260. This is a small correlation value and it has very little significance. The Mongoose client's correlation for 0.5% loss is -0.5133, it has a probd value of 1.8456×10^{-40} and a probrs value of 2.3318×10^{-46}. This value is larger and is significant, this loss level has the largest difference between the correlations.

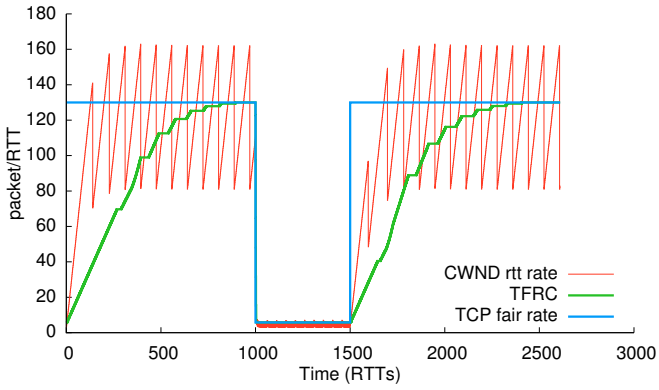
Difference is the difference between the throttle value and the TCP fair value. The difference value is larger in the Second Life than in Mongoose, meaning that Mongoose is producing a value that is more accurate. The 0.5% loss rate difference values are 1,137.78 kbps for Second Life and 29.94 kbps for Mongoose. This is the largest difference, as it is less than the Second Life client's adaptation threshold.

Summary graphs The median throttle values for different loss levels has been graphed for the Second Life client and the Mongoose client. The graphs show the distribution of throttles for a range of different loss levels.

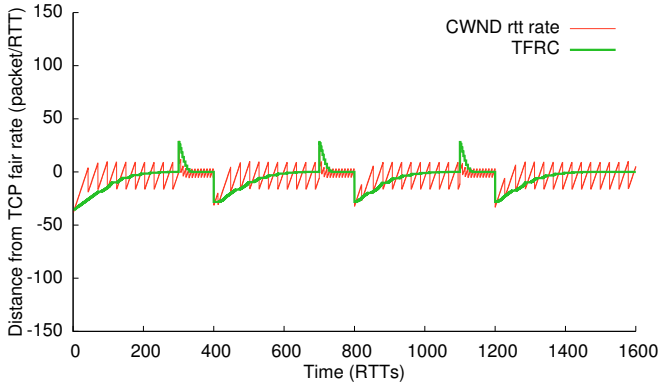
The mean of the values calculated by the client and sent to the server by the Mongoose client are shown in figure 7.5a. The x-axis shows the fraction of the packets that are lost. The y-axis shows the sending rate in Kbps. The line is the TCP fair equation for a RTT of 150 ms and a packet size of 376 bytes. This is a significant improvement



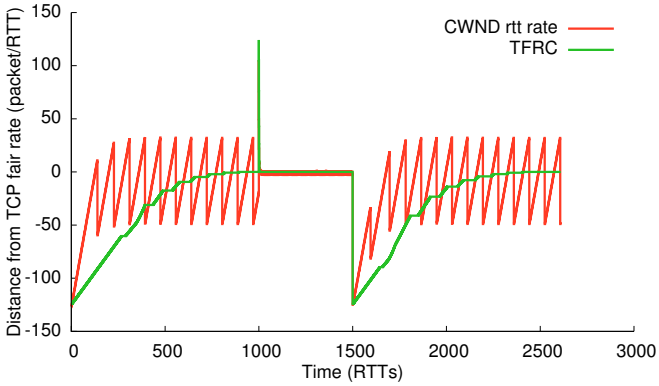
(a) 0.1% loss and 1% loss



(b) 0.01% loss and 5% loss



(c) 0.1% loss and 1% loss Distance from the TCP fair rate.



(d) 0.01% loss and 5% loss Distance from the TCP fair rate.

Figure 7.7: Simulations of the behaviour types of Congestion control system with different loss rates.

over the value in figure 7.5b which shows the Second Life calculated throttle values from the Second Life client. The Mongoose client is able to calculate and send throttle values that are TCP fair for a range of loss rates. The Second Life client is not adapting to network conditions until the loss rate is over 2% and is not producing TCP fair values.

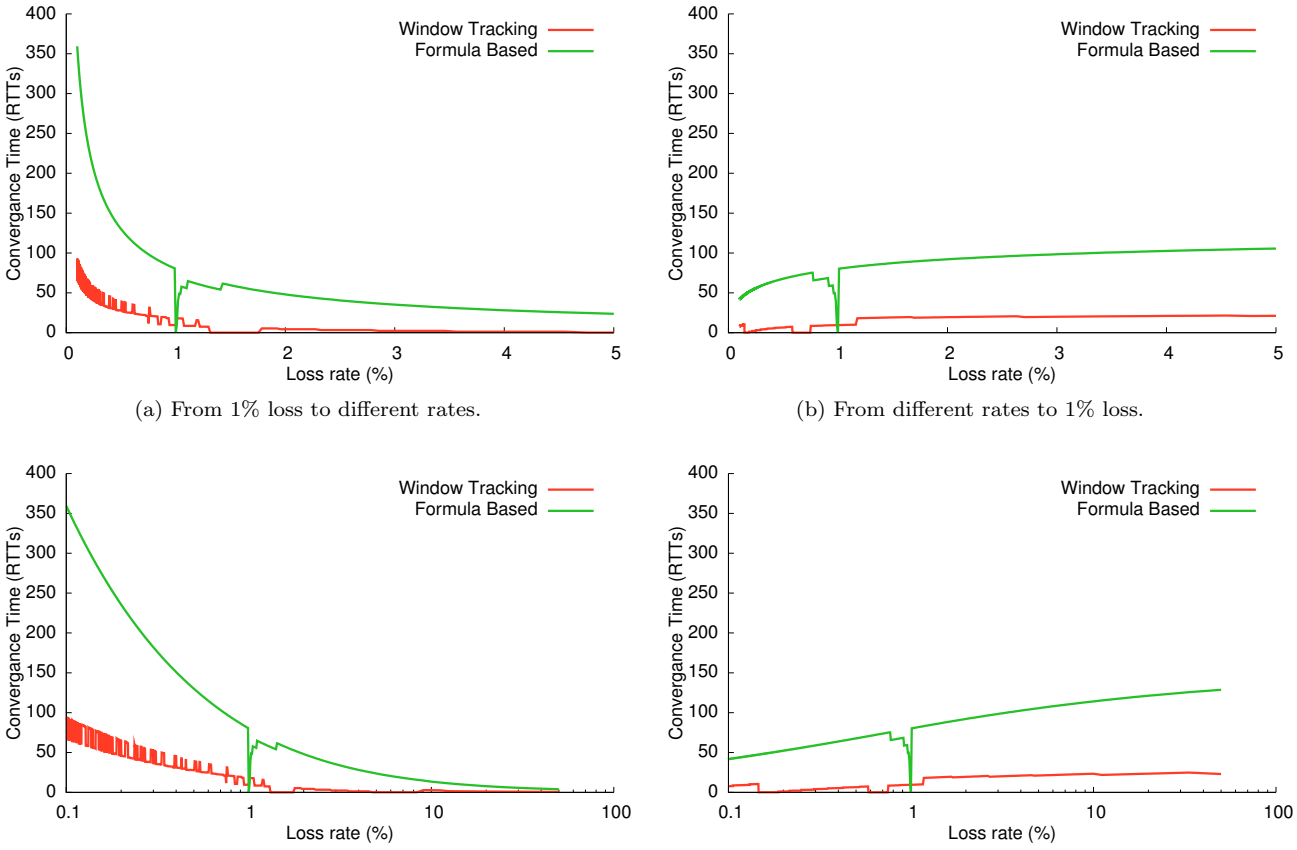
The Second Life client produces throttle values that are not TCP fair. When its loss rate is over the loss level that causes it to reduce the throttle value changes in the loss rate do not affect the throttle value. The Mongoose throttle values are close to the TCP fair values. The Mongoose client increases its throttle value in response to decreases in the loss rate. The throttle values used by both clients has a minimum and maximum values which mean that they will only produce correct values in between a range of values. At zero loss both clients are calculating, as expected, a value that is the maximum value. The window tracking algorithm in the Mongoose client is able to produce average throttle values that are TCP fair.

7.3 Comparison of the Mongoose window tracking system with TFRC.

The Mongoose client has to produce a rate to send to the client, to work with the Second Life servers. The most common type of rate based congestion control uses equation based rate calculation. This is because the majority of rate based system are proposed are for applications such as video and audio that require a stable rate. Second Life does not require a stable rate.

In this section the difference between equation based and window tracking congestion control will be examined. The distribution of TFRC values calculated from the Mongoose traces against the mean loss levels are examined. Simulations of equation and window tracking system are examined. These simulations are compared to real traces.

The median TFRC values calculated from the Mongoose traces, are shown in figure 7.6a. The x-axis is the loss rate and the y-axis is the sending rate in kbps. The red points are the median and 25 and 75 percentiles of the calculated TFRC values. The green line shows the TCP fair value the same as in figure 7.5a. The upper and lower percentiles are closer together for the TFRC values than for the throttle values calculated by Mongoose, this is expected and



(a) From 1% loss to different rates.

(b) From different rates to 1% loss.

(c) From 1% loss to different rates in log-scale.

(d) To different rates to 1% loss in log-scale.

Figure 7.8: Convergence time for Window tracking and formula based congestion control in simulations.

consistent with the simulated values.

The median values for the window tracking system are similar to the TFRC values, both are close to the average values from the TCP fair equation. There is more to TCP fairness than the average bandwidth usage. The rate of convergence is important, quick convergence is better than slow.

Simulations of a window tracking and formula based systems has been performed with the loss rate changed between different loss rates, a simulation of connections where the loss rates was varied between different values is shown in figure 7.7. The x-axis gives the time in round trip times (RTTs) and the y-axis gives the bandwidth utilization in RTTs. The red line is the window tracking system and the green line shows the formal based system.

(a) Connections where the loss rate is varied between 0.1% and 1% are shown in figure 7.7a. The loss rate remains at 0.1% for 200 RTTs and then goes to 1% for 100 RTTs, this repeats three times and then it goes to 0.1% loss for the rest of the simulation.

(b) Connections where the loss rate is varied between 0.01% and 5% are shown in figure 7.7b. The loss rate stays at 0.01% for 1000 RTTs and then goes to 5% for 500 RTTs and then returns to 0.01%. The formula based system takes longer to reach its peak value than the window tracking system.

(c) The distance between the calculated rate and the TCP fair rate were the loss rate is varied between 0.1% and 1% are shown in figure 7.7c. This is the same simulation as in figure 7.7a. The distance increases when the loss rate changes because it takes time for the calculated rate to change.

(d) The distance between the calculated rate and the TCP fair rate were the loss rate is varied between 0.1% and 1% are shown in figure 7.7d. This is the same simulation as in figure 7.7b.

The time that window tracking and formula based systems take to converge is different, figure 7.8 show the convergence time. The x-axis shows the loss rate and the y-axis shows the time taken to converge to the correct value. The red line is the window tracking system and the green line shows the formal based system.

(a) The time to converge to a number of different value from 1% loss is shown in figure 7.8a. The x-axis in this figure shows the loss rate that has been changed to. As the loss rate increases the convergence time decreases. The window tracking system converges more quickly than the formula system.

(b) The time to converge from a number of different loss rate to 1% loss is shown in figure 7.8b. The x-axis in this figure shows the loss rate that has been changed from. As the starting loss rate increases the time to reach the correct rate increases.

(c) The time to converge to a number of different values from 1% loss is shown in figure 7.8c, on a log scale.

(d) The time to converge from a number of different loss rates to 1% loss is shown in figure 7.8d, on a log scale.

The changes in the loss rate happen for both the rate calculation systems. The formula system increases and decreases more slowly than the window tracking system. Window tracking systems change their value quickly moving around the mean throttle value. The mean value of the window tracking rate changes more quickly than the formula throttle system, it quickly settles into the correct pattern. In these simulations the higher loss rate parts are shorter than the lower loss rate parts.

Traces have been recorded were the loss rate was changed during the trace, three of the four graphs in figure 7.6 show these traces. The x-axis shows the time in seconds since the beginning of the trace. The y-axis shows the bandwidth in kbps. The red line shows the calculated throttle values, the green line shows the TFRC value calculated from the trace and the blue line shows the TCP fair loss rate calculated from the loss rate extracted from the traces.

(b) A trace where the loss rate was changed from 0.1% to 1%, is shown in figure 7.6b. The window tracking system changes more quickly than the TFRC values. The mean loss rate for the entire trace is 0.79%, the median loss rate is 0.44%, the 25 percentile is 0% and the 75 percentile is 1.30%. The median throttle value is 302.90 Kbps and the average TCP fair value is 287.27 Kbps giving a ratio of 1.05. The median TFRC value is 320.63 Kbps, this gives a ratio between this and the TCP fair rate of 1.12.

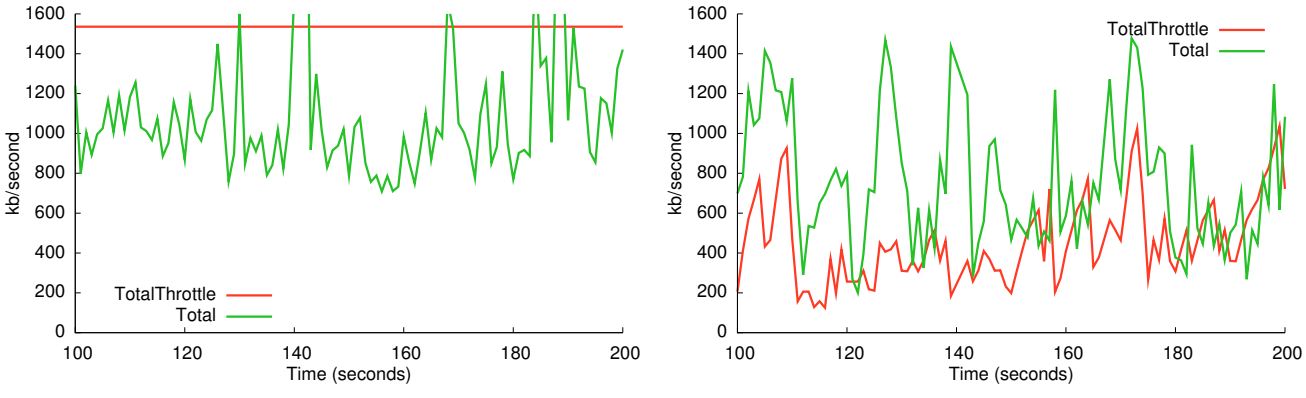
(c) A trace where the loss rate was changed from 0.01% and 5%, is shown in figure 7.6c. When the loss rate changes the TCP fair changes first, then the Mongoose window tracking system and then the TFRC value. Once the values have changed to the new rate they are close together. The traces mean loss rate is 4.44%, the median loss rate is 4.35%, the 25 percentile is 2.34% and the 75 percentile is 6.25%. The median throttle value is 152.85 Kbps and the average TCP fair value is 120.81 Kbps giving a ratio of 1.27. The median TFRC value is 164.63 Kbps, this gives a ratio between this and the TCP fair rate of 1.36.

(d) A trace where the loss rate was changed from 5% to 0.05%, is shown in figure 7.6d. When the loss rate changes the window tracking system gets in front then the TFRC value and the TCP fair value last. After a little while the TCP fair value gets ahead of the other value and reaches its correct value first and then the window tracking system. The mean loss rate for the trace is 0.93%, the median loss rate is 0% and the maximum loss rate is 14.71%.

$$X = \frac{c}{RTT \cdot p^d} \tag{7.2}$$

The TFRC formula can be written as is shown in equation 7.2 (from [190]). Where c and d are constants, which differ between different congestion control algorithms. For standard TCP $d = 0.5$. A larger d value causes the congestion control system to increase more quickly to its stable rate, a larger value also increases the unfairness between connections with different RTT values. The Scalable TCP (STCP) [93] system has a d value of 1 and has a constant amount of time, in RTTs between the peaks in its window value. This makes STCP able to work in long RTT links but make it unable to compete between connections with different RTT values.

The TFRC values are not clearly better than the Mongoose values. The window tracking system adapts to the changes in loss level more quickly than equation based. The window tracking system will be used in Mongoose and the TFRC system will not be used.



(a) 0.01% loss.

(b) 0.5% loss.

(c) 3% loss.

(d) 10% loss.

Figure 7.9: Mongoose calculated throttle values and bandwidth utilization with different loss levels.

7.4 Bandwidth utilization against requested bandwidth.

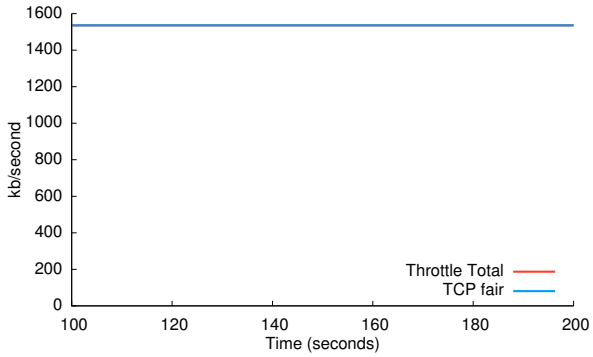
The window tracking system used in Mongoose has been compared with the bandwidth used. The responsiveness of the server to the throttle values sent from the client has been investigated. The effect of the mongoose client without an interval between its throttle calculations. The calculated throttle values and the affect of this client on the bandwidth utilization has been investigated. The difference different intervals have on the effectiveness of the throttle system is investigated in this section.

Bandwidth utilization and throttle values. The throttle values and bandwidth utilization against time for a connection in a busy environment where the avatar was walking and the Mongoose client was used from 100 seconds to 200 seconds are shown in figure 7.9. In each graph the x-axis gives the time in seconds and the y-axis gives the bandwidth in kbps. The red line is the transmitted throttle value, sampled once a second. The green line shows the number of bits per second sent from the server to the client. These are the same traces as in figure 7.4.

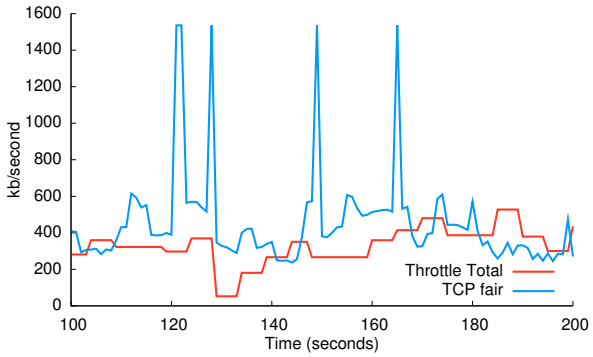
(a) The bandwidth and throttle values for a loss rate of 0.01% are shown in figure 7.9a. The bandwidth utilization changes allot throughout the trace, the throttle value doesn't change much. When the throttle values are decreasing the clients throttle values appears to decrease though it is not contained by the throttle values. The median throttle value is 1,500 kbps and the median bandwidth utilization is 829.52 kbps giving a ratio of 0.55.

(b) The bandwidth and throttle values for a loss rate of 0.5% is shown in figure 7.9b. The peaks are lower than the 0.01% loss graph. The bandwidth goes up and down throughout the trace, when the throttle value decreases so does the bandwidth. The median throttle value is 391.42 kbps and the median bandwidth utilization is 546.78 kbps giving a ratio of 1.40.

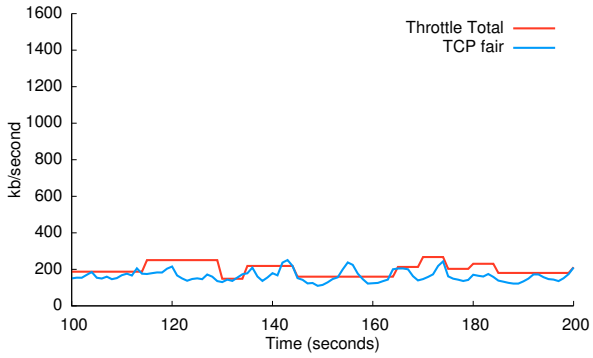
(c) The bandwidth and throttle values for a loss rate of 3% is shown in figure 7.9c. The median throttle value is 144.43 kbps and the median bandwidth utilization is 311.35 kbps giving a ratio of 2.16.



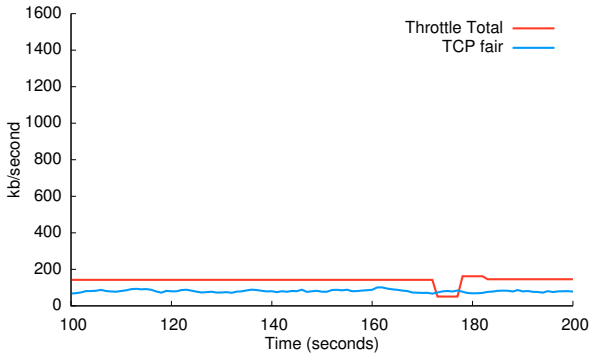
(a) Bandwidth utilization and TFRC values with 0.01% loss.



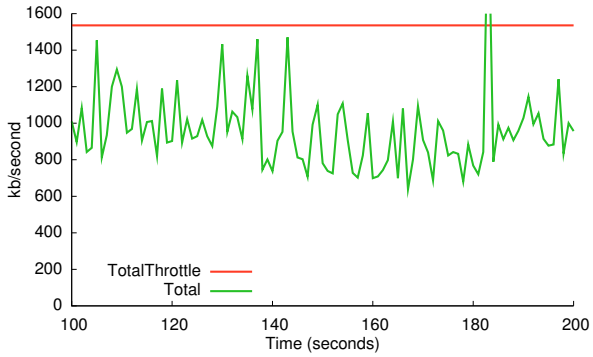
(b) Bandwidth utilization and TFRC values with 0.5% loss.



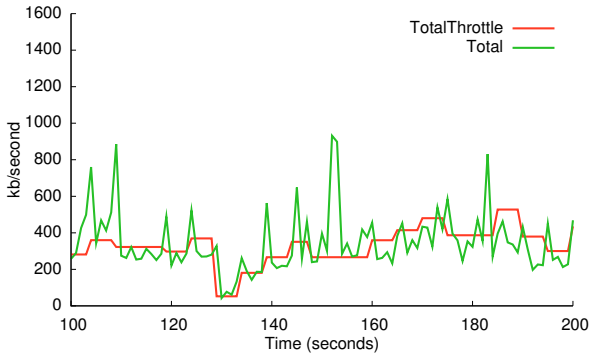
(c) Bandwidth utilization and TFRC values with 3% loss.



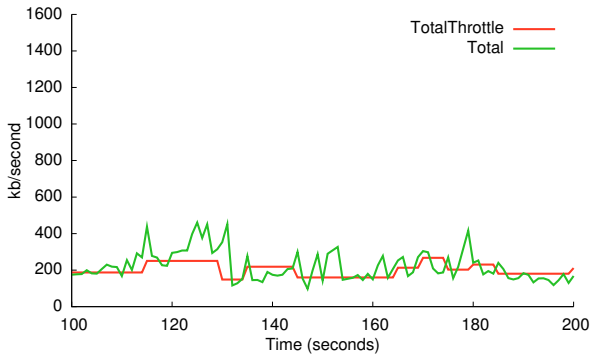
(d) Bandwidth utilization and TFRC values with 10% loss.



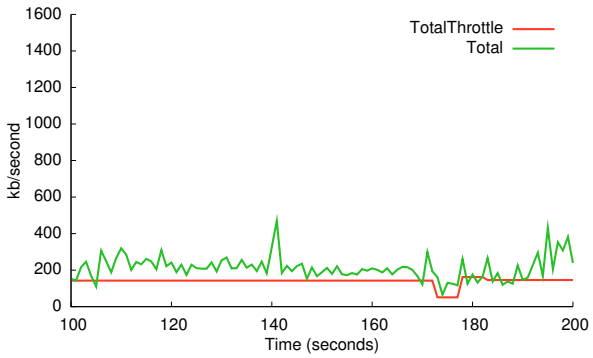
(e) Calculated throttle values and bandwidth utilization with 0.01% loss.



(f) Calculated throttle values and bandwidth utilization with 0.5% loss.

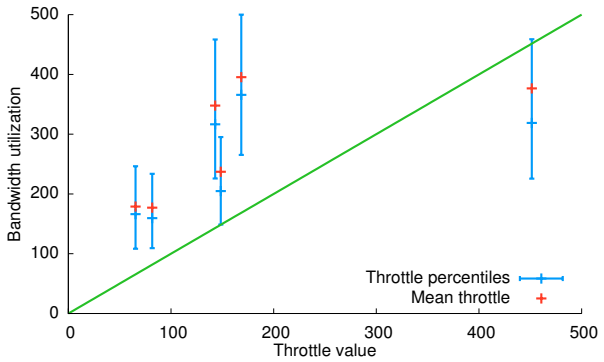


(g) Calculated throttle values and bandwidth utilization with 3% loss.

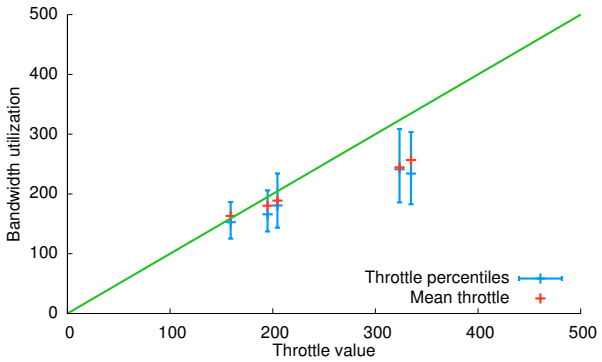


(h) Calculated throttle values and bandwidth utilization with 10% loss.

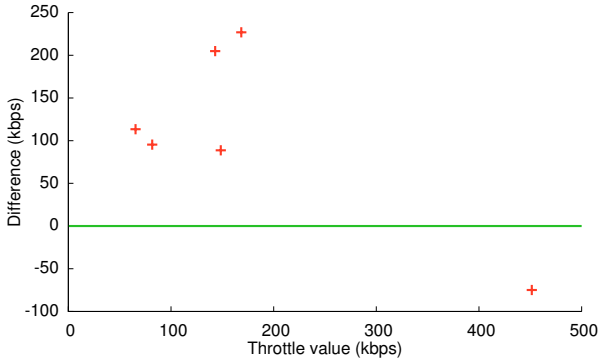
Figure 7.10: Mongoose calculated throttle values, bandwidth utilization and TFRC values with different loss levels and a 5 second calculation interval.



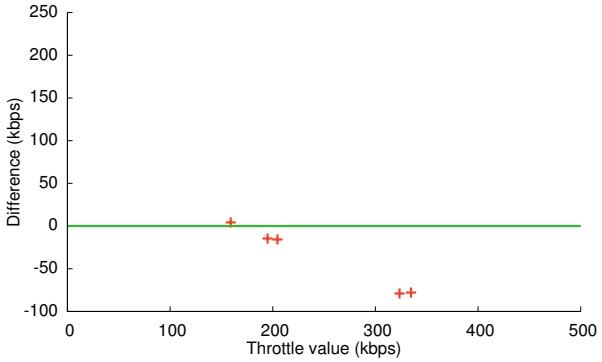
(a) Requested bandwidth against bandwidth for no rate calculation interval.



(b) Requested bandwidth against bandwidth for 5 second calculation interval.



(c) The difference between bandwidth and the throttle for no rate calculation interval.



(d) The difference between bandwidth and the throttle for 5 second calculation interval.

Figure 7.11: Requested bandwidth against bandwidth and the difference between bandwidth and the throttle.

(d) The bandwidth and throttle values for a loss rate of 10% is shown in figure 7.9d. The median throttle value is 80.18 kbps and the median bandwidth utilization is 283.31 kbps giving a ratio of 3.53.

The peaks are lower as the loss rate increases. A lower throttle value sent by the client results in a lower sending rate but the sending rate does not closely follow the value sent by the client. This is the opposite of the Second Life client that does not calculate a TCP fair sending rate but is able to control the sending rate. The affect of the throttle seems less than in earlier graphs.

In summary a version of the client without an interval in the calculation was created that produces rate calculations that are close to TCP fair. This client was not very affective at controlling the rate of packet transmission. The Second Life client has an interval between sending throttle values to the server. We hypothesise that the interval between the new throttle values being sent to the server is important to the ability of the client to control the bandwidth used by the server.

7.4.1 Calculation interval

A client with an interval of 5 seconds between the throttle calculations being performed has been created in order to determine if the rate of throttle updates sent to the server is important.

Throttle calculation: Time series The calculated values were extracted from traces with different levels of loss, figure 7.10 shows the time series graphs for four different levels of loss, they are the first four. In each graph the x-axis gives the time in seconds and the y-axis gives the bandwidth in kbps. The red line is the transmitted throttle values extracted from the trace, sampled once per second. Both the throttle value and the TFRC values are limited to 1,500 Kbps. The values are in a similar range. This version of the client produces a calculated value that is less close to the TFRC value than the Mongoose client with no calculation interval though it is still in the correct range.

(a) The throttle values for a loss rate of 0.01% are shown in figure 7.10a. The mean loss rate is 0.013%, the mean loss rate is 0% and the maximum loss rate is 2.2%. The throttle value does not change from the maximum value. The median throttle value is 1,500 kbps and the average TCP fair value is 1,500 kbps giving a ratio of 1.

(b) The throttle values for a loss rate of 0.5% are shown in figure 7.10b. The mean loss rate is 0.58%, the median loss rate is 0% and the maximum loss rate is 12.5%. The throttle values are closer to the TCP fair value than the Second Life values and less close than the values in figure 7.4b. The median throttle value is 333.29 kbps and the average TCP fair value is 335.71 kbps giving a ratio of 0.99.

(c) The throttle values for a loss rate of 3% are shown in figure 7.10c. The mean loss rate is 3.17%, the median loss rate is 2.7%, the 25 percentile is 1.5%, the 75 percentile is 4.7% and the maximum is 16.7%. The median throttle value is 199.50 kbps and the average TCP fair value is 142.96 kbps giving a ratio of 1.40.

(d) The throttle values for a loss rate of 10% are shown in figure 7.10d. The mean loss rate is 11.32%, the median loss rate is 10.5%, the 25 percentile is 7.7%, the 75 percentile is 14.3% and the maximum is 36.4%. The median throttle value is 128.07 kbps and the average TCP fair value is 75.67 kbps giving a ratio of 1.69.

The client is changing the value more slowly than the TFRC value and also has a minimum change in value that it will send to the server which results in the stepping feature seen in the graph. The TFRC value is reacting more slowly than throttle system.

Throttle calculation: Summary Statistics The mean of the values calculated by the client and sent to the server by the Mongoose client with a calculation interval of 5 seconds are shown in figure 7.12a. The x-axis shows the fraction of the packets that are lost. The y-axis shows the sending rate in Kbps. The line is the TCP fair equation for a RTT of 150 ms and a packet size of 376 bytes. This is a significant improvement over the value in figure 7.5b which shows the calculated throttle values from the Second Life client. The Mongoose client is able to calculate and send throttle values that are TCP fair for a range of loss rates. The Second Life client is not adapting to network conditions until the loss rate is over 2% and is not producing TCP fair values.

The throttle values calculated by this client with a 5 second interval are not as good as the calculations made by the client without a calculation interval. The calculated values are still better than the Second Life values, the produced are values are still close to the TCP fair values.

Requested bandwidth against bandwidth utilization. The throttle value and bandwidth utilization against time for a connection in a busy environment were the avatar was walking and the experimental client was used from 100 seconds to 200 seconds are shown in figure 7.10, they are the last four. In each graph the x-axis gives the time in seconds and the y-axis gives the bandwidth in kbps. The red line is the transmitted throttle value, sampled once a second. The red line is the transmitted throttle values extracted from the trace, sampled once a second. The green line shows the number of bits per second sent from the server to the client.

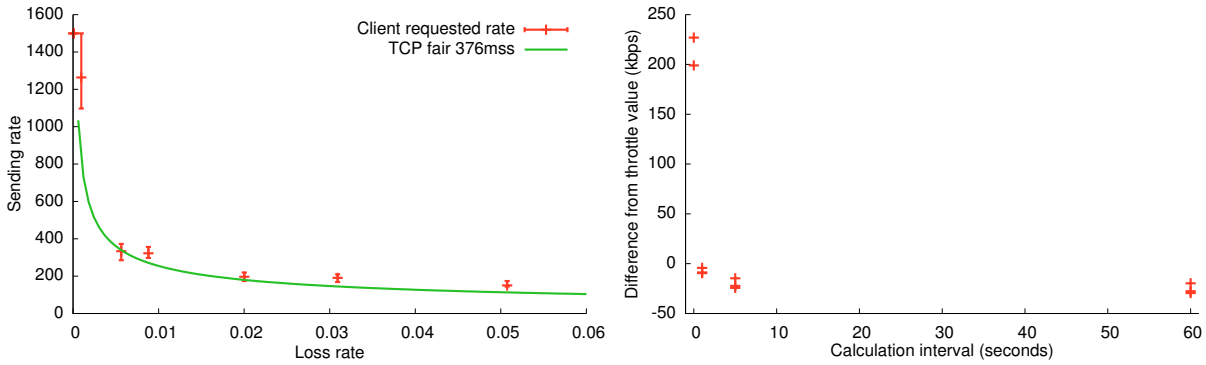
(e) The bandwidth and throttle values for a loss rate of 0.01% are shown in figure 7.10e. The bandwidth utilization changes allot throughout the trace, the throttle value doesn't change.

(f) The bandwidth and throttle values for a loss rate of 0.5% is shown in figure 7.10f. The peaks are lower than the 0.01% loss graph. The bandwidth goes up and down throughout the trace, when the throttle value decreases so does the bandwidth. The bandwidth spikes above the throttle line, the throttle value spends most of its time at or below the throttle value.

(g) The bandwidth and throttle values for a loss rate of 3% are shown in figure 7.10g. The peaks are lower than for lower loss rates, the bandwidth spends more of its time over the throttle value.

(h) The bandwidth and throttle values for a loss rate of 10% are shown in figure 7.10h. The bandwidth usage spends more of its time exceeding the throttle value than in the other traces, work in chapter 5 has shown that there is a lower bound to the amount of bandwidth Second Life will use.

The Mongoose client with a 5 second calculation interval is able to control the bandwidth utilization of the server better than the client without a calculation interval. The used bandwidth tracks the throttle value more closely than for the client with no interval.



(a) Mongoose with 5 second calculation interval.

(b) Difference between the requested bandwidth and used bandwidth against the calculation interval.

Figure 7.12: Difference between the requested bandwidth and used bandwidth against the calculation interval.

Comparison between calculation intervals The value of the throttle sent from the client against the bandwidth used is shown in the top half of figure 7.11. The x-axis shows the throttle value sent and the y-axis shows the amount of bandwidth used. The green line is $y = x$, where the values are expected to fall. We hypnosis that Second Life is application and network limited in its bandwidth usage. If the client is successfully limiting the traffic the value will be at or below the line. For larger throttle values the throttle will be less of a limiting factor as the application will be the limiting factor.

(a) The values for the client without a calculation interval are shown in figure 7.11a. There are several values above the line meaning the throttle system is unable to limit the bandwidth utilization to the set value.

(b) The values for the client with a five second calculation interval are shown in figure 7.11b. The values are mostly below the line.

As the avatar moves around the island the objects and textures that need to be sent to the client. The throttle can therefore change the amount of traffic, overall, that needs to be sent to the client. This also means that whilst the changes to the throttle affect the quiet and busy section of the traffic, a low throttle value it does result in the total bandwidth usage being equal to the throttle value for the entire trace.

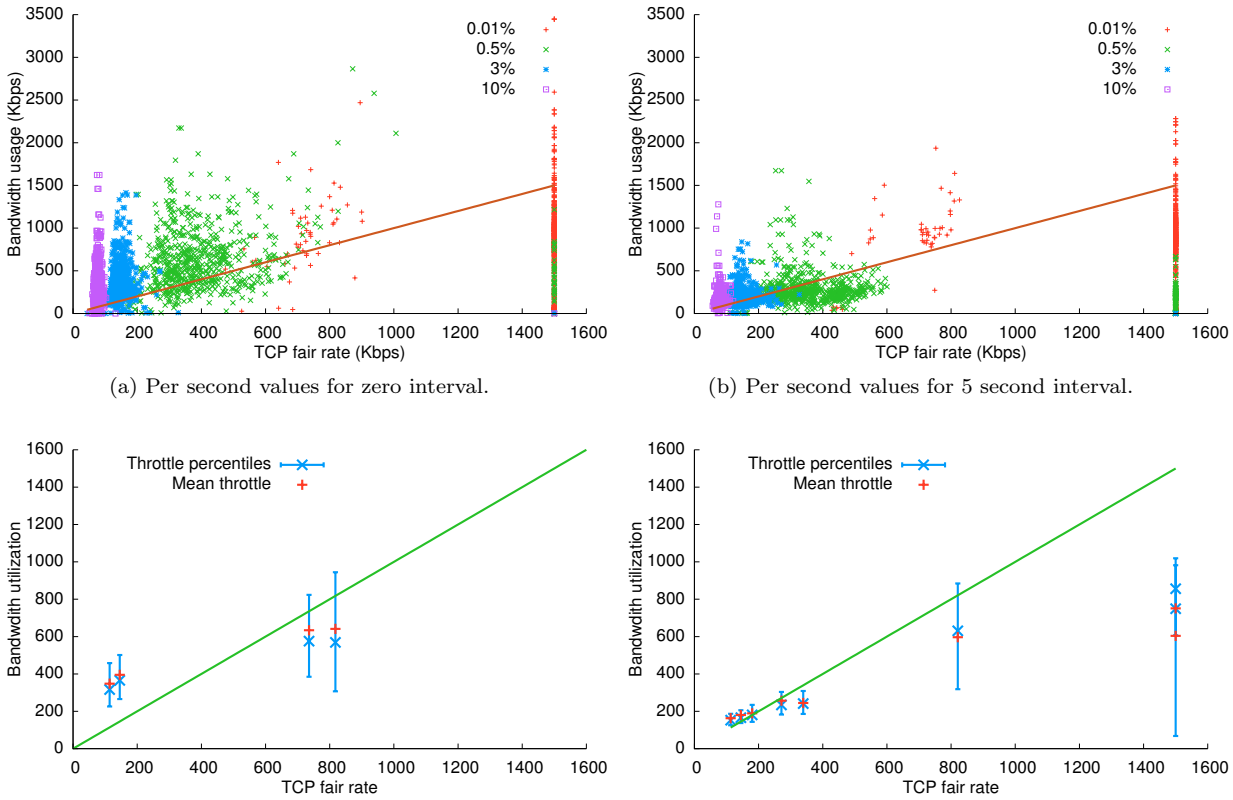
(c) The difference between the throttle value and the bandwidth used against the throttle value for the interval less client are shown in figure 7.11c.

(d) The same thing for the client with a 5 second calculation interval are shown in figure 7.11d.

Without a calculation interval the value sent from the client has less affect on the bandwidth utilization of the server. At low bandwidths the bandwidth used by the server exceeds the requested bandwidth for the interval less client.

Difference against calculation rates. The difference between the calculated bandwidth and the used bandwidth against the calculation interval is show in figure 7.12b. The x-axis gives the shows the interval between performing throttle calculations. The y-axis gives the difference between the sent throttle value and the amount of bandwidth used. The value shown are for a loss rate of 3%. The calculation intervals used were 0, 1, 5 and 60 seconds. A positive value indicates that the bandwidth utilization is larger than the throttle value. The values for an interval of zero is positive and large, so the system is not being very successful in controlling the bandwidth utilization. When there is an interval the difference is negative and small.

It is important to have an interval between the calculations of the throttle value and sending it to the server. When there is no interval the client is not able to control the bandwidth used by the server.



(a) Per second values for zero interval.

(b) Per second values for 5 second interval.

(c) Average values for zero interval.

(d) Average values for 5 second interval.

Figure 7.13: Bandwidth usage against TCP fair value.

7.5 Bandwidth utilization and TCP fair rate.

The TCP fairness of the bandwidth utilization of the Mongoose client has been measured. The different versions of the Mongoose client with different intervals have been examined and the median values compared to those of the standard Second Life client.

Bandwidth utilization against TCP fair rate. The bandwidth against the TCP fair values for different Mongoose traces are shown in figure 7.13. The top two are per second from the four of the traces and the bottom two show the mean, median and percentile value for different traces. The x-axis is the TCP fair rate and the y-axis is the Bandwidth usage in Kbps. In the top two graphs the red points are for 0.01% loss, the green points are for 0.5% loss, the blue points are for 3% loss and the purple points are for 10% loss. The line is for $y = x$.

(a) The per second TCP fair against bandwidth for the Mongoose client with no interval between transmissions of throttle values is shown in figure 7.13a. A lot of the points of the same colour are grouped together and then there is a group at the end were the TCP fair value has its maximum value. A large number of the points are above the line.

(b) The per second TCP fair against bandwidth for the Mongoose client with a 5 second interval is shown in figure 7.13b. Less of the points are above the line than in the first graph.

In the per connection graphs in figure 7.13d, the red points are the mean bandwidth values, the blue error bars are the median and percentiles and the green line is $y = x$.

(c) The per connection TCP fair against bandwidth for the Mongoose client with no interval between transmissions of throttle values is shown in figure 7.13c. At low throttle values the bandwidth values are above the TCP fair values.

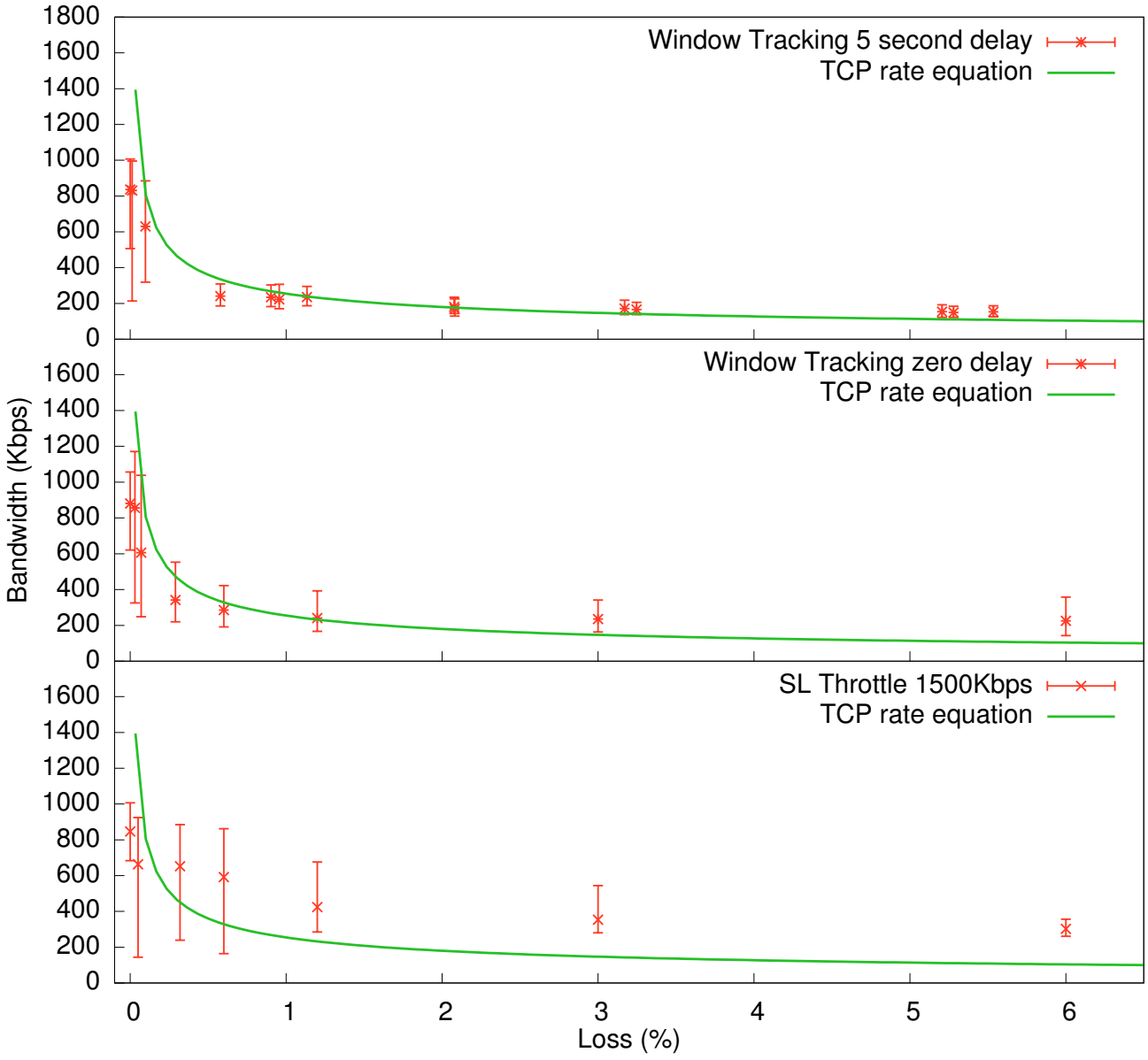


Figure 7.14: Bandwidth against loss rate.

(d) The per connection TCP fair values against bandwidth values for the Mongoose client with a 5 second interval is shown in figure 7.13d. The low value are closer to the line than in the other graph. The client with a sending interval is able to keep the bandwidth within the TCP fair range.

The TCP fair throttle values calculated by the client is able to produce a sending rate that is TCP fair.

Bandwidth utilization against Loss rate. The bandwidth against loss rate for the Mongoose and Second Life clients is shown in figure 7.14. The Second Life client's throttle value was set to an initial value of 1,500 Kbps. The points are box plots are of the median, the 25 percentile and the 75 percentile of bandwidth utilization. The green line is the TCP fair equation value.

Bottom The values for the Second Life client are shown in the bottom graph. At very low loss levels the Second Life client is TCP fair. The rate is not TCP fair for loss rates over 0.5% it is not TCP fair.

Middle The middle graph shows the values for the Mongoose client with no calculation interval. This version is closer to the TCP fair line, with the median values not being over the line until 3% loss. The values over the line are closer to the line than the values for the Second Life client.

Top The top graph shows the Mongoose client with a five second calculation interval. The values are closer to the line than the values for the zero interval client. The percentile values are closer to the mean value than in the other too graphs, except at the maximum values. The values for over 3% loss are still over the line, but they are much closer to the line than the interval-less client.

This demonstrates that it is possible to increase the adaptiveness of SL so that it can take advantage of the improved performance available in high bandwidth low congestion environments and automatically adjust to a lower bandwidth utilisation in the presence of congestion. This is achieved, without the need to alter the servers at all.

7.6 Conclusion

The methodology was to perform experiments with real connections connected to the Second Life servers across the Internet and to change the network conditions and record the traffic.

The throttle values calculated by the Second Life client and the Mongoose client were compared with TCP fair and TFRC. The Second Life values were not equal to the TCP fair values, as they make no attempt to be. The Second Life values are also above the TCP fair values in several different circumstances. The Mongoose values are close to the TCP fair values. The Correlation between the throttle value and the loss was calculated for both clients. The Second Life clients had very little correlation in the wrong direction. The Mongoose client had more correlation in the correct direction.

The window tracking system used by Mongoose was compared with TFRC. The median TFRC values are close to the TCP fair values for the average levels of loss. Simulations of window tracking systems and equation based congestion control systems were performed. The window tracking systems converges on the correct values more quickly than the equation based system. Traces of Mongoose traffic, with its window tracking system confirm these results. The window tracking system is therefore appropriate for Mongoose.

The affect of the throttle value on the amount of bandwidth used by the Second Life server was investigated. The Mongoose client with no interval between sending new throttle values to the server had only limited success in controlling the bandwidth utilization. A client with an interval of 5 seconds was tested. The calculated throttle values were compared to TCP fair, the values were a little less close to TCP fair than the zero interval client but still close to TCP fair. The bandwidth utilization are close to the throttle values or they are lower. Different intervals were tested and 5 seconds seems like a reasonable interval.

The TCP fairness of the bandwidth utilization that the Mongoose client results in was tested and found to be in the correct range.

In Summary: The Mongoose client is able to react to loss and calculate a rate that is TCP fair. The Mongoose client is able to control the bandwidth used by the server. The version of Mongoose with no interval is able to calculate a rate correctly but is unable to control the bandwidth used by the server. The Mongoose client with a five second interval in the calculation of the throttle value calculates a throttle value that is TCP fair and is able to control the bandwidth.

Chapter 8

Traffic Management with OpenSim

This chapter looks at traffic management for OpenSim a toolkit for creating virtual worlds that are compatible with Second Life servers. As OpenSim servers are freely available and open source this facilitates investigation of both the client and server side aspects of traffic management for virtual worlds. This in turn allows the constraint that only changes to the client are investigated, to be relaxed. It is reasonable to relax this constraint as it is possible for Linden Labs to implement modifications to Second Life's [115] server and there is increasing use of OpenSim. Using OpenSim [132] we were able to run, measure, modify and evaluate a virtual world server. In doing so it was possible to both investigate how a client and server can co-operate together to ensure TCP [160] fair [146] behaviour as well as exploring how network bandwidth should be distributed between different traffic types within the application.

Original contributions in this chapter include: a study of OpenSim network traffic, the design and evaluation of TCP fair congestion control for OpenSim and server side traffic control which both adapts to requests from the client for aggregate bandwidth and distributes that bandwidth between traffic types. Real-time measurements are used to determine network conditions and these measurements are used as parameters to traffic management algorithms. The client measures packet loss and RTT to determine a TCP fair rate which is requested from the server. The server side system combines two level token buckets with proportional fair queuing to regulate data transmissions.

The OpenSim server uses the same network protocol as Second Life and provides a virtual environment that is similar to that provided by Second Life. Although OpenSim uses the same protocol as Second Life the bandwidth requirements and traffic mix may differ substantially. This chapter opens with a measurement study of OpenSim traffic, this looks at the bandwidth required for a range of common activities, bandwidth requirements for different OpenSim "islands" and packet size distributions. The OpenSim server can be managed and administered by institutions like other types of server. There are also facilities for modifying the server, new modules providing additional functionality may be developed and added. From a development stand point OpenSim has the advantage that it is open-source so can be modified, as can the Second Life client (SLC) [112].

The Mongoose client was created to control the bandwidth usage of the Second Life server. Here its interaction with the OpenSim server is evaluated. In the light of this evaluation modifications to the OpenSim server are proposed, implemented and evaluated. This chapter is organised into the following sections:

Section 8.1: The design priorities of an OpenSim traffic management system.

Section 8.2: A measurement study of the traffic from OpenSim.

Section 8.3: The examination of OpenSim traffic and throttle system.

Section 8.4: A description of OpenSim and the design and implementation of the modified OpenSim throttle system.

Section 8.5: The measurements of the bandwidth and throttle values with the modified OpenSim server.

Section 8.6: A comparison of the two client versions and the two server versions.

8.1 Traffic management system

There are a number of metrics that measure the quality of service that a connection receives from the network. The throughput is the amount of data transferred per unit time. The rate of packet loss is also important, it will affect the throughput. Network delay is also important as it affects the responsiveness of the system.

Environments such as Virtual Worlds have similar timeliness requirements to 3D games [1][2][3]. In addition when the world has to remain consistent the input often has to be sent and response received before the user input has an effect on the environment. As a result there are limits on the level delay and variation in the delay that can be tolerated by the user. In MUVEs such as Second Life [115] the user can move around in an environment even when only some of the information, necessary to render the environment, has been transferred to the client.

In Second Life and OpenSim traffic is separated into different channels each channel has its own queue, and there is a scheduling system which controls when the packets in each queue are sent. These scheduling algorithms attempt to achieve the desired weighting between the different queue, without having a large detrimental effect on the performance of the system. A fair queueing system attempts to maximise the minimum bandwidth that each of the flows, or channels, is able to achieve. If the packet size is taken into account then flows with small packets get an equal opportunity to transmitter data.

A system to control OpenSim's traffic should react to network conditions by changing the rate at which packets are transmitted. It should be TCP fair. The existing system involves the client determining the rate at which the server is to send data and sending this information to the server, which then caps its data transfer rate. The client side is the same system that Second Life uses, the server is different. In previous chapters modifications to the client were proposed which involved modifying the client to detect loss and measure the round trip time and uses these values to calculating a TCP fair sending rate, which is then requested as a cap.

The current channel system separates different types of data into channels, each channel receives a separate bandwidth allocation. However, the channels are not arranged by the Quality of Service requirements of the traffic. Modifying the client and the server makes it possible to change the channel system so that QoS requirements are better met.

There are three main types of data, avatar control traffic, data which defines the shape of objects and data which defines the surfaces of objects. In addition there is data which defines the land, trees and wind. If a user's avatar does not respond to input promptly the interactivity and immersiveness of the system will be reduced. The bandwidth necessary for avatar traffic is low, a few kilobits per second for each avatar. Thus avatar traffic has a low bandwidth requirement and high requirement for timeliness. This suggests that it should be prioritised. Prim traffic defines the shape or structure of objects. This traffic has a higher bandwidth requirement than avatar traffic, but is not as critical for interactivity. Texture traffic, consists of images which are used to provide realistic surfaces for objects. Textures have a higher bandwidth requirement, but a lower priority than prim traffic.

In the current channel structure the Task channel contains the information about avatars and information about Prims. Images are dealt with by a separate Texture channel. There is a Resend channel, which contains all of the re-sent packets, this results in Texture, Land, Asset and Task packets getting mixed together. The different types of packet then cannot be given different amounts of bandwidth when it comes to resending them. An improved channel system would separate the Avatar and Prim traffic:

1. Asset: Containing the contents of the Asset channel.
2. Avatar: Containing all of the avatar traffic.
3. Task: Containing the remainder of the Task channel.
4. Layer: Containing the Wind, Land and Cloud channels.
5. Texture: Containing the contents of the Texture channel.

With the re-sent packets belonging to the same channel as the original sending of the packets. With the avatar traffic having a separate channel it will be possible to guarantee it bandwidth. A channel system implementation based upon this design and compatible with the current SL protocol is described in sub section 8.4.1.

8.2 OpenSim traffic examination

These experiments develop an investigation of the bandwidth requirements and composition of OpenSim traffic for a range of avatar activities and for different virtual world spaces. This section is organised into the following subsections:

[1] Epic Games and Digital Extremes. Unreal Tournament 2004. Atari `http://www.unrealtournament2003.com/ut2004/`, 16 March 2004. [Online; accessed 28-March-2011]

[2] id Software. Quake 3 arena. `http://www.quake3arena.com/`, 2 December 1999. [Online; accessed 29-September-2010]

[3] RavenSoftware. Quake 4, 2005. `http://www.idsoftware.com/games/quake/quake4/` [Online; accessed 30-June-2011]

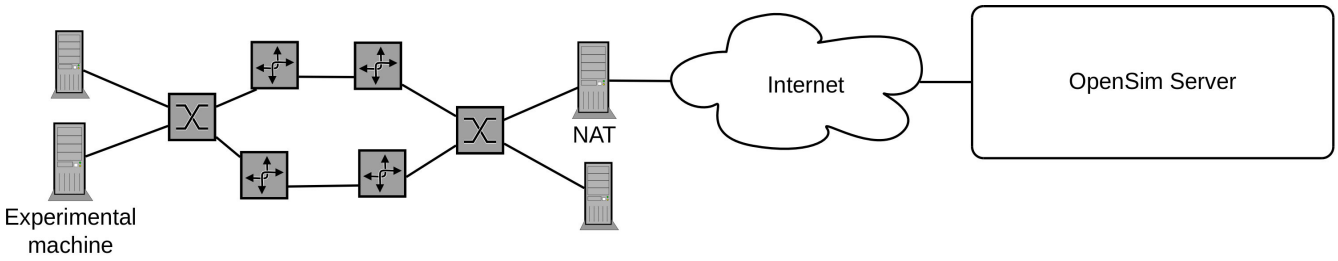


Figure 8.1: The structure of the test network.



(a) Sparta (b) Cathedral4 (c) Aeolus (Wifi Island)

Figure 8.2: Screen shots of the three islands used.

- The methodology and experimental design is described.
- The bandwidth required for the main avatar activities is measured. This includes standing, walking, teleporting and flying.
- Traces representing traffic from three different islands are presented.
- A decomposition of traces by packet size is presented. This includes identifying the main packet types and giving the distribution of packet sizes.
- How packets and bytes are distributed between the channels are analysed.

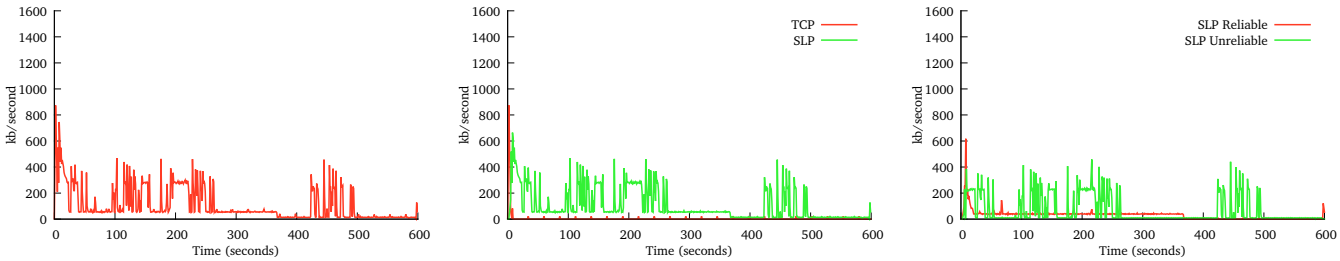
8.2.1 Methodology

A series of controlled experiments creating real connections across a test network to OpenSim servers. The traffic that was received and sent from the Second Life client was captured for post processing. The avatar was put through the activities of walking, teleporting, flying and standing. The primary activity undertaken was walking along a set path. During the experiments the network conditions were measured to provide context for the work. Each scenario lasts for ten minutes.

The experiments were carried out at the University of St Andrews School of Computer Science. Traffic statistics were generated for the global traffic and for individual traffic categories of interest. The results are presented tabularly and through time and distribution graphs. The client's cache was emptied between each run.

Connections were fed through a router, which was equipped with the netem network emulator [73]. This facilitates control over available bandwidth, levels and patterns of loss and delay. It is described in more detail in section 3.3.

The netem system was used to introduce extra delay and loss into the network. The network used allowed loss to be introduced in both directions. Loss levels of 0%, 1%, 3% and 10%, were used and delay of 120 ms was introduced. These loss levels are different from those used for evaluation of Second Life server where a minimum loss level of 0.01%, which was found to have to effect on the Second Life or Mongoose clients. The traffic was captured at both sides of the traffic shaping network, this allows the introduced loss rate per unit time to be calculated accurately. The resulting RTT can be calculated from the captured traffic. This allows a TCP fair rate to be calculated and to be compared against the rate value sent from the client and against the rate at which the server sends packets to the client.



(a) The total number of bytes a second.

(b) The number bytes a second for each protocol.

(c) The reliable and unreliable bytes per second against time.

Figure 8.3: The total amount of packet and bytes sent from an avatar walking around the cathedral island.

The islands that were used in these experiments are on two different OpenSim servers. Three islands were used, they are called Sparta, Cathedral and Aeolus. Screen shots of an avatar standing on these three islands are shown in figure 8.2. Sparta contains a reconstruction of the ruins of a Spartan basilica, this is based upon a real excavation of a Byzantine Spartan Basilica [187]. This allows students to explore this Basilica and its surrounding environment, in its current state. The surrounding environment takes up four islands with the basilica being on the island called Sparta. The Sparta island has 5233 primitives. The basilica is missing a large amount of parts including the roof and a lot of the walls. There are hills and a large number of trees. There are other structures and paths.

The cathedral4 island contains a reconstruction of the cathedral in St Andrews. The reconstruction is spread over 6 islands, island number 4 has the largest number of prims on it. The Cathedral4 island has 18326 primitives.

Aeolus is a teaching tool that performs WiFi [39] simulations, it is an instantiation of the WiFi Virtual Laboratory (WifiVL) island [184]. The Aeolus island has 1521 primitives. A large portion of traffic from islands consists of the textures so the number of prims does not directly relate to the amount of bandwidth used. Aeolus island is a learning tool about Wifi. It contains slides, videos and simulations.

This island contains a central structure surrounded by different features explaining WiFi. It contains a museum with large posters explaining the history of wireless communication. It then contains posters and simulations explaining different scenarios in wireless communication. There is an interactive text based explanation of the packet format. There is a set of slide shows explaining different wireless technologies and the standard documents. The island allows the users to create simulations of WiFi networks.

TCPDump[4] was used to capture the packets. The SLparse, described in section 3.2, which we created for analysing Second Life traffic was used in the post processing.

The time series graphs in this chapter show the entire length of the trace, except where user wise state, to show the change in behaviour of the client sever system over length of the trace.

In summary real connections are created to an OpenSim server and a set actions are performed in world. The traffic is captured and the network conditions controlled.

8.2.2 Summary statistics and analysis by protocol of an OpenSim session.

Summary statistics from a connection where an avatar walks around the cathedral Island for ten minutes are shown in table 8.1. It shows bytes packets and connections separated into TCP and SLP. Most of the traffic is User Datagram Protocol (UDP) [156] packets forming a single connection. All of the packets in this connection were delivered in

[4]V. Jacobson, C. Leres, S. McCanne, et al. Tcpdump. `http://www.tcpdump.org/`, 1989. [Online; accessed 03-November-2010]

	Total	**SLP**	**TCP**
KB	8,023.60	7,813.87	209.73
Packets	23,066	21,839	1,227
Kbps	106.98	104.19	2.80
Connections	153	6	147

Table 8.1: Summary statistics for a trace with the avatar walking around the cathedral island.

order without gaps. This trace is similar to the Second Life traces, however the bandwidth usage is less than from the islands used in the Second Life evaluation chapter.

The number of bytes per second for the entire trace overall and per protocol are shown in figure 8.3. At the beginning of the trace there are a number of TCP connections. These TCP connections are the client downloading the welcome web page. The client then logs on to the system using Hypertext Transfer Protocol Secure [54, 166] (HTTPS). The circuit is then created. The x-axis shows the time in seconds since the beginning of the trace.

(a) The total number of bytes per second against time for the trace is show in figure 8.3a. The y-axis is bandwidth in kbps. The first spike in bandwidth is the HTTPS data, the next spike is the prims in the environment that do not move. The mean bandwidth is 110 kbps and the median bandwidth is 56.8 kbps, the maximum bandwidth is 895 kbps, the 25 percentile is 16.0 kbps and 75 percentile is 174 kbps. The traffic is bursty showing that it is application limited

(b) The number of bytes with each protocol against time is shown in figure 8.3b. The y-axis is bandwidth in kbps. There is very little TCP traffic and it is concentrated in short connections. The mean TCP bandwidth is 2.87 kbps, the median value is 0 kbps and maximum value is 895 kbps. The mean of the OpenSim packets is 107 kbps, the median value is 56.4 kbps, the 25 percentile is 15.3 kbps and the 75 percentile is 159 kbps.

(c) The number of reliable and unreliable bytes per second against time for the trace is show in figure 8.3c. The y-axis is bandwidth in kbps. The mean bandwidth of reliable packets is 3.67 kbps, the median is 4.80 kbps, the maximum is 79.1 kbps, the 25 percentile is 0.2 kbps and the 75 percentile 5.14 kbps. The mean bandwidth of unreliable packets is 8.75 kbps, the median value is 1.26 kbps, the maximum value is 58.7 kbps, the 25 percentile is 1.14 kbps and 75 percentile is 12.0 kbps. The balance of bandwidth between reliable and unreliable SLP packets is different from those from Second Life as shown in figure 5.5b. The ratio between the bandwidth usage from reliable and unreliable is 0.42 and for the SLLab trace it is 3.16.

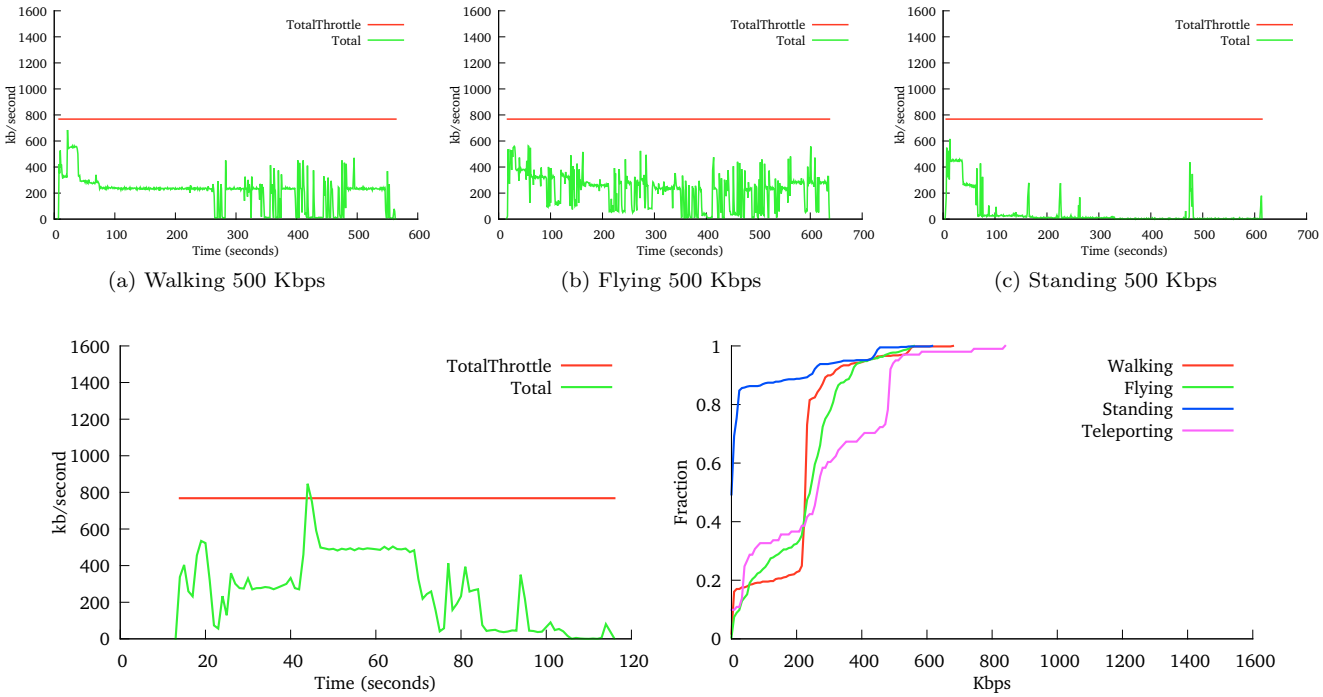
There is a background level of packets and bytes from the avatar walking around the island plus more packets when the avatar encounter things in the environment. The overall bandwidth usage is larger than that of multi player network games but smaller than Second Life. The bandwidth is below the default throttle value sent from the client. There are more reliable packets sent at the beginning of the connection than unreliable. The number of unreliable packets then increases and the number of reliable packets decrease. The much larger difference between the ratio of bytes of reliable and unreliable than packets means that average sizes of the reliable and unreliable packets is different between Second Life and OpenSim. The difference in the amount of bandwidth may be caused by the difference between the islands there may also be a deference because of the designs of OpenSim and Second Life servers. As the bandwidth usage of OpenSim is smaller the default throttle value was used instead of the maximum value so that there would be limitation by the throttle under low loss levels.

8.2.3 Standing, Walking, Flying and Teleporting

To determine the effect of in world activities on the traffic, the throttle values the bandwidth utilization for traces under different in world activities on an island called Sparta, with the throttle value set to its default of 500 kbps, are shown in figure 8.4. The x-axis shows the amount of time in seconds since the beginning trace and the y-axis shows the bandwidth in kilobits per second (Kbps). The red line shows the throttle value and the green line shows the bandwidth usage.

(a) The values for a session where the client is walking around an island with the throttle value set to 500 Kbps is shown in figure 8.4a. The bandwidth usage is below the throttle value. The throttle value stays at the same level after it is initially sent. The mean bandwidth is 214 kbps, the median is 234 kbps, the maximum is 684 kbps, the 25 percentile is 224 kbps and the 75 percentile 241 kbps. The traffic is throttled with a 31.0% of the samples at 232 kbps and 14% of the samples at 8 kbps. The mean and the median are below the throttle value.

(b) The values for a session where the client is flying around an island with the throttle value set to 500 Kbps is shown in figure 8.4b. The mean bandwidth is 224 kbps, the median is 249 kbps, the maximum is 562 kbps, the 25 percentile is 114 kbps and the 75 percentile is 296 kbps. The bandwidth usage when the avatar is flying a little higher than when the avatar is walking. At large number of the samples are at a similar level.



(a) Walking 500 Kbps (b) Flying 500 Kbps (c) Standing 500 Kbps

(d) Teleporting 500 Kbps (e) CDF of SWFT

Figure 8.4: The bandwidth usage and throttle values for different activities on the Sparta island.

(c) The values for a session where the client is standing in the island with the throttle value set to 500 kbps is shown in figure 8.4c. The bandwidth usage is small. The mean bandwidth is 51.0 kbps, the median is 8.09 kbps, the maximum is 616 kbps, the 25 percentile is 1.66 kbps, the 75 percentile is 23.8 kbps. The shape of the initial bandwidth usage is similar to the shape of the same part of the trace for the walking though the bandwidth usage is lower.

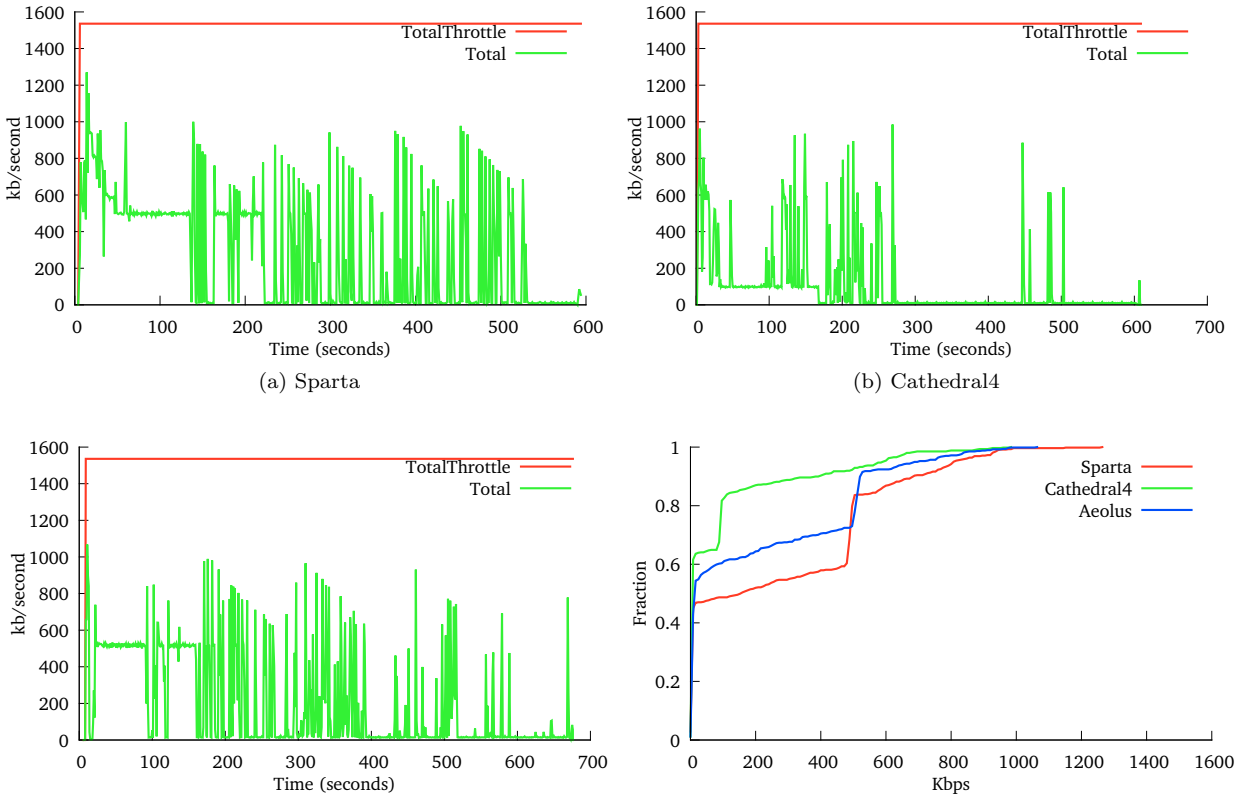
(d) The values for a session where the client is teleporting between islands with the throttle value set 500 kbps is show in figure 8.4d. The mean bandwidth is 269 kbps, the median is 271 kbps, the maximum is 847 kbps, the 25 percentile is 48.4 kbps, the 75 percentile is 484 kbps. This trace is shorter than the others as the traffic reduces after affect of teleporting finished the traffic would be same as standing.

(e) The CDF of the bandwidth from these traces are shown in figure 8.4e. The x-axis shows the bandwidth in Kbps and the y-axis shows the fraction of the total that are less than or equal to this level of bandwidth. The red line is the values for the avatar walking, the green line is for the avatar flying, the blue line in for the avatar standing and the pink line is for the avatar teleporting. The walking line shows that there are a large number of samples with the same of similar values, the flying line also shows this though it is less clear, this could indicate that they are limited to this level. The teleporting line also shows evidence of throttling though at a higher level.

The bandwidth usage is less than the throttle value when in all of these traces, the OpenSim throttle system seems more able to control the amount of bandwidth used. The bandwidth usage for teleporting is largest followed by flying, then walking and finally standing. The bandwidth usage is more stable than in Second Life traces and it is less bursty. The bandwidth looks shaped and the walking and flying traffic are around the same level as they are limited. The teleport bandwidth starts high and then reduces as the avatar is not doing anything else.

The traffic is similar to that of Second Life in the amount of bandwidth and mix of transport protocols. The bandwidth is more than that of multiplayer networked games [186, 24, 186][5]. There are regions of application limitation during the traces. The bandwidth is quite stable and less than the throttle value. The bandwidth is less than that of Second Life.

[5]Valve Software. Counter-Strike: A Counter-Terrorism Half-Life Modification. `http://www.counter-strike.net/`, 5 July 1995. [Online; accessed 03-October-2010].



(a) Sparta

(b) Cathedral4

(c) Aeolus (Wifi Island)

(d) CDF of the bandwidth for the islands.

Figure 8.5: The throttle and bandwidth usage for walking on different islands.

8.2.4 Islands

The bandwidth against time for three islands used in this experiment are shown in figure 8.5. The x-axis shows the time in seconds since the beginning of the trace and the y-axis shows the bandwidth in kbps. The red line shows the throttle value sent from the client to the server. The green line shows the amount of user data sent in OpenSim packets from the server to the client.

(a) The bandwidth and throttle value for the Sparta island is shown in figure 8.5a. The throttle value is 1536 kbps. The mean bandwidth value is 281 kbps, the median is 161 kbps, the maximum is 1270 kbps, the 25 percentile is 9.94 kbps and the 75 percentile is 500 kbps. The bandwidth appears to be throttled for parts of the trace. The bursts are larger and shorter than the same island with a lower throttle value.

(b) The bandwidth and throttle value for the Cathedral4 island are shown in figure 8.5b. The throttle value is 1536 kbps. The mean bandwidth value is 101 kbps, the median is 10.8 kbps, the maximum is 985 kbps, the 25 percentile is 9.29 kbps and the 75 percentile is 98.6 kbps. The texture bandwidth for this island comes in short burst and is smaller than the other traces.

Island	Activity	Throttle	Mean	25%	Median	75%	Maximum
Sparta	**Walking**	500	214	224	234	241	684
Sparta	**Flying**	500	224	114	249	296	562
Sparta	**Standing**	500	51.0	1.66	8.09	23.8	616
Sparta	**Teleporting**	500	269	48.4	271	484	847
Sparta	**Walking**	1,500	281	9.94	161	500	1,270
Cathedral4	**Walking**	1,500	101	9.29	10.8	98.6	985
Aeolus	**Walking**	1,500	208	14.4	19.2	508	1,069

Table 8.2: Summary of bandwidth levels for the different traces.

(c) The bandwidth and throttle value for the Aeolus island are shown in figure 8.5c. The throttle value is 1536 kbps. The mean bandwidth value is 208 kbps, the median is 19.2 kbps, the maximum is 1068 kbps, the 25 percentile is 14.4 kbps and the 75 percentile is 508 kbps.

(d) The CDF for the bandwidth from the traces from different islands are shown in figure 8.5d. The x-axis shows the bandwidth in Kbps and the y-axis shows the fraction of the total that are less than or equal to this level of bandwidth. The red line shows the values for the Sparta island, the green line shows the values for the Cathedral4 island and the blue line shows the Aeolus island. There are a large number of samples for the Sparta island at the same level, this level is higher than for the traces where the throttle value is smaller The Sparta and Aeolus islands are throttled at a higher level than from the walking traces with a lower throttle level.

The bandwidth for the Sparta and Aeolus islands appear to be throttled and to be using more bandwidth than the cathedral island. The cathedral island has a large number of prims which are not spread out and there are not as many textures. The throttle values have been increased to their highest value. This allows the maximum amount of bandwidth that could be generated by the islands. With the throttle value set higher the bandwidth usage is higher and less stable with bursts. The cathedral4 island uses less bandwidth than the other islands, despite the large number of prims on the island. The Sparta and Aeolus islands have similar bandwidth usage though the Sparta island is higher.

The summary statistics for the bandwidth values from the SWFT and islands traces are shown in table 8.2. The SWFT traces are for the Sparta island with the throttle set to its default value of 500 kbps and the island trace are for the avatar walking with the throttle value set to its maximum value of 1500 kbps. The bandwidth for flying is large and more variable than the bandwidth for walking. Increasing the throttle value from 500 kbps to 1500 kbps for walking on the Sparta island has increased the mean bandwidth and decreased the median value. When the traffic is less throttled a small amount of extra data is sent and the uses of bandwidth happens in shorter bursts.

8.2.5 Packet sizes

The distribution of packet sizes is shown in figure 8.6. This is from the same trace as figure 8.3, from the cathedral island. The height of each part represent the portion of the packets that are of that type and that size or smaller. The x-axis shows the packet size in bytes. The y-axis shows the fraction of the total.

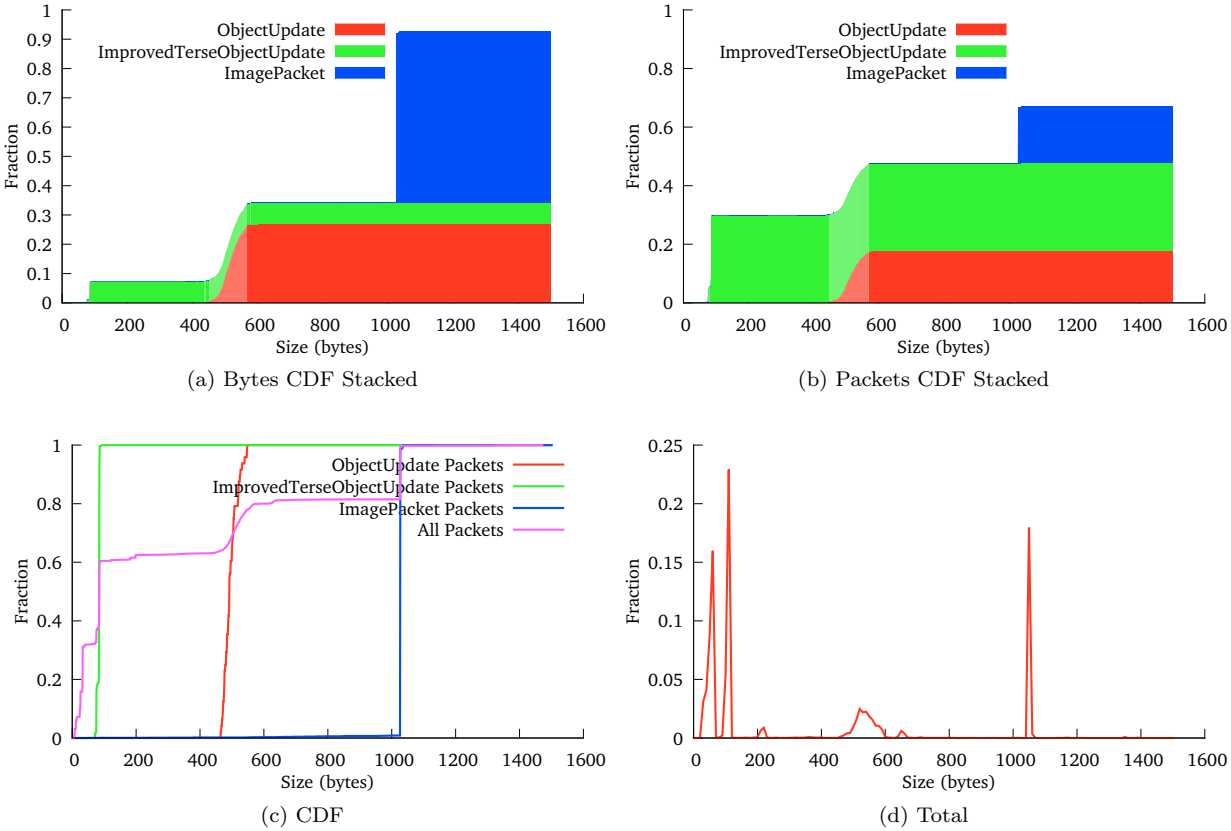
(a) The cumulative distribution frequency (CDF) for the packets [113][6] that make up more than 2% of the total bandwidth, staked on top of one another are show in figure 8.6a. The height of each part represents the amount of the data that is in packets of that type and that size or smaller. The majority of `ImagePacket` bytes are in packets 1027 bytes in size (98%). The majority `ImprovedTerseObjectUpdate` bytes are in packet 85 bytes in size (67.9%). The minimum size of `ObjectUpdate` packets 465 bytes and the maximum size is 548 bytes. 92% of the data is in packets of these three types.

(b) The portion of packets that are of a certain size are shown in figure 8.6b. The size of each part represent the portion of the packets that are of that type and that size or smaller. The data is mostly in the three packet types though there are a large number of packets that are not of these types, 67% of the packets are of these types.

(c) The number of bytes that are in packets smaller than a size for the 3 types are show in figure 8.6c. The red line shows the `ObjectUpdate` packets, the green line shows the `ImprovedTerseObjectUpdate` packets and the blue line shows the `ImagePacket` packets. The pink line shows the cumulative distribution frequency of the packet sizes for all of the packet types. It shows the changes in the number packets that size or smaller lining up with larger changes in packets of that size or smaller for the different packet types.

(d) The number of packets of each size for the entire connection is shows in figure 8.6d. It shows that a large number of the packets are less than 200 bytes. It shows that at least 40% of the bytes are in packets smaller than 200 bytes and a large portion of the remaining data belongs in packets larger than 1000 bytes. The packets that aren't of the three most common types are small.

[6]Second Life: packet type templates `https://svn.secondlife.com/svn/linden/sandbox/2007/VWR-2972_1.18.2/scripts/messages/message_template.msg` [Online; accessed 10-November-2010]



(a) Bytes CDF Stacked

(b) Packets CDF Stacked

(c) CDF

(d) Total

Figure 8.6: The distribution of packet sizes for the cathedral trace.

This shows that the majority of the data is in three types of packets. These packets are of each mostly of the same size. The majority of the rest of the data is made up from `ImageData`, `CoarseLocationUpdate` and `LayerData` packets. `ImageData` packet are large but there are not many of them. `CoarseLocationUpdate` packets are small but there are a large. There are two types of `LayerData` packet, wind and land.

The majority of the data is in packets of these three types, with 92% of the packets being of these types. The majority of the packets are also of these types, with 67% being of these types. These packets mostly of the same size, the `ImagePacket` packets are almost all of the same size. The `ImprovedTerseObjectUpdate` and `ObjectUpdate` packets are of a similar size but not all of the same size. The number of different packets used is less than that are used by Second Life.

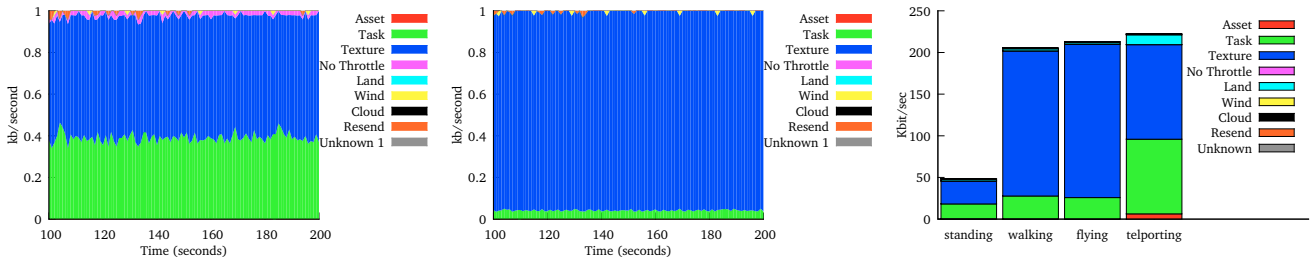
8.2.6 Channels

The share of the bandwidth taken by each channel [113] for an avatar walking around the Sparta island is shown in figure 8.7. The red part is the Asset channel, the green part is the Task channel, the dark blue part is the Texture channel, the pink part is traffic not in any channel and the light blue part is the land channel. These time series graphs show the time from 100 seconds to 200 seconds to show the steady state of the system.

(a) The channel portions against time are shown in figure 8.7a. The x-axis is the time in seconds since the beginning of the trace, the y-axis is the fraction of the total packets in each channel. The majority of the packets are Task and Texture, with around 40% for the Task channel and around 60% for the Texture throttle.

(b) The portion of the bandwidth taken by each channel against time are shown in figure 8.7b. The majority of the bytes are from the Texture channel, for this trace it is 84.74%. The majority of the rest is from the Task channel 13.28% for the entire trace.

(c) The average amounts of bandwidth used by each channel for different activity are shown in figure 8.7c. The standing trace has less bandwidth usage and far less Texture packets. This is not surprising as the avatar is not



(a) Time series packets. (b) Time series bytes. (c) Total values

Figure 8.7: Fraction of bandwidth per throttle for the Sparta island.

moving and so not encountering as many objects with textures. The walking, flying and teleporting traces have around the same amount of average bandwidth. The Teleporting trace last less long than the others because after the teleportation has finished it would be the same as the standing trace.

The task packets make up large portion of the packets but they are small so they make up a small portion of the data. The mix in throttle usage is stable throughout this trace, this is more stable than in Second Life traces. Some of the channel use very little bandwidth, most of the traffic is from the texture channel. The allocation from the client is that the Task and Texture channels only get less than 30% of the bandwidth. This is described in section 4.3.

8.2.7 Summary

There are a larger number of packets at the beginning of the trace than at the end, these packets are from the state of the objects on the island. The bandwidth is more tightly throttled in OpenSim than it is in Second Life. The activity under taken in world affects the traffic and amount of traffic. Different islands with different numbers and types of objects create different amounts and patterns of traffic. There is a small amount of TCP traffic in the traces, with several small connections. The SLP packets can be marked reliable or not, the unreliable packets are more common in OpenSim traffic than the reliable. The difference in numbers between the reliable and unreliable are larger than in Second Life.

The traffic consists mostly of three packet types, `ObjectUpdate`, `ImprovedTerseObjectUpdate` and `ImagePacket`. The packets are of a small number of packet sizes. The `ObjectUpdate` packets in the Task channel and `ImagePacket` packets are in the Texture channel. The majority of the packets and bytes are from the Texture channel.

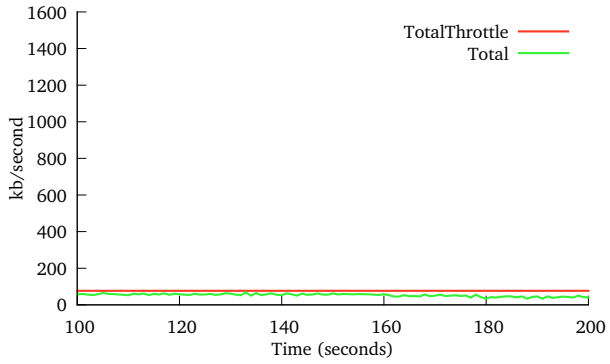
The bandwidth is more stable than the traffic from Second Life. The bandwidth is less because the throttle limits the bandwidth to a lower level and there may also be less objects and textures in the environment than there would be in Second Life. The mix of packet types is different from in Second Life traffic including that there are less SLP reliable packets. The data is mostly from the Task and Texture channels. Which is not consistent with the allocations to the channels by the Second Life client.

8.3 Throttle system and network conditions.

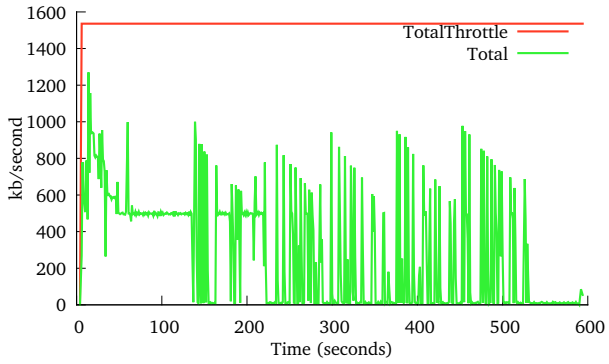
This section investigates the behaviour of the OpenSim throttle system. The affect of throttle settings and different loss rates were investigated. This section first looks at the affects of changing the throttle setting on the bandwidth usage and distribution. The effect of loss on the throttle value calculated by the SLC and the amount of bandwidth used is then investigated. The effect of loss on the throttle value calculated by the Mongoose client and the amount of bandwidth used by the server is investigated. The effect of the individual throttle on the overall bandwidth is investigated.

8.3.1 Second Life client and throttle settings.

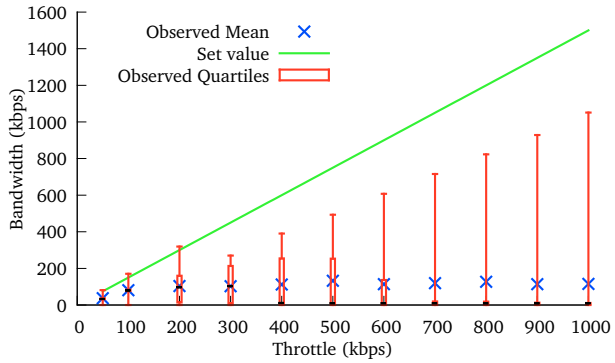
The traffic was captured for a range of settings and network conditions was investigated. The affect of the throttle setting at the client on the bandwidth usage. The mean, median, maximum, minimum, and upper and lower quartiles are show in figure 8.8. The x-axis shows the set throttle value in kbps. The y-axis shows the bandwidth usage in kbps.



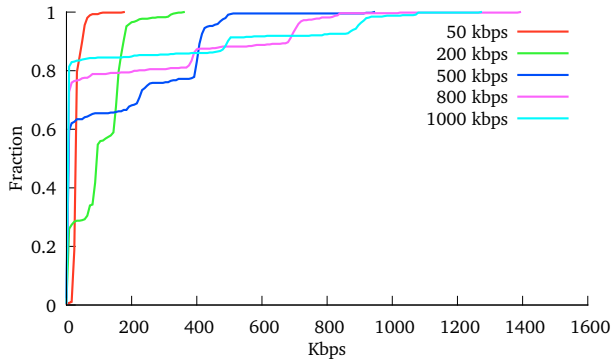
(a) Walking Sparta 50 Kbps



(b) Walking Sparta 1500 Kbps



(c) Bandwidth against throttle values.



(d) CDF of bandwidth against throttle values.

Figure 8.8: Bandwidth usage against Throttle values.

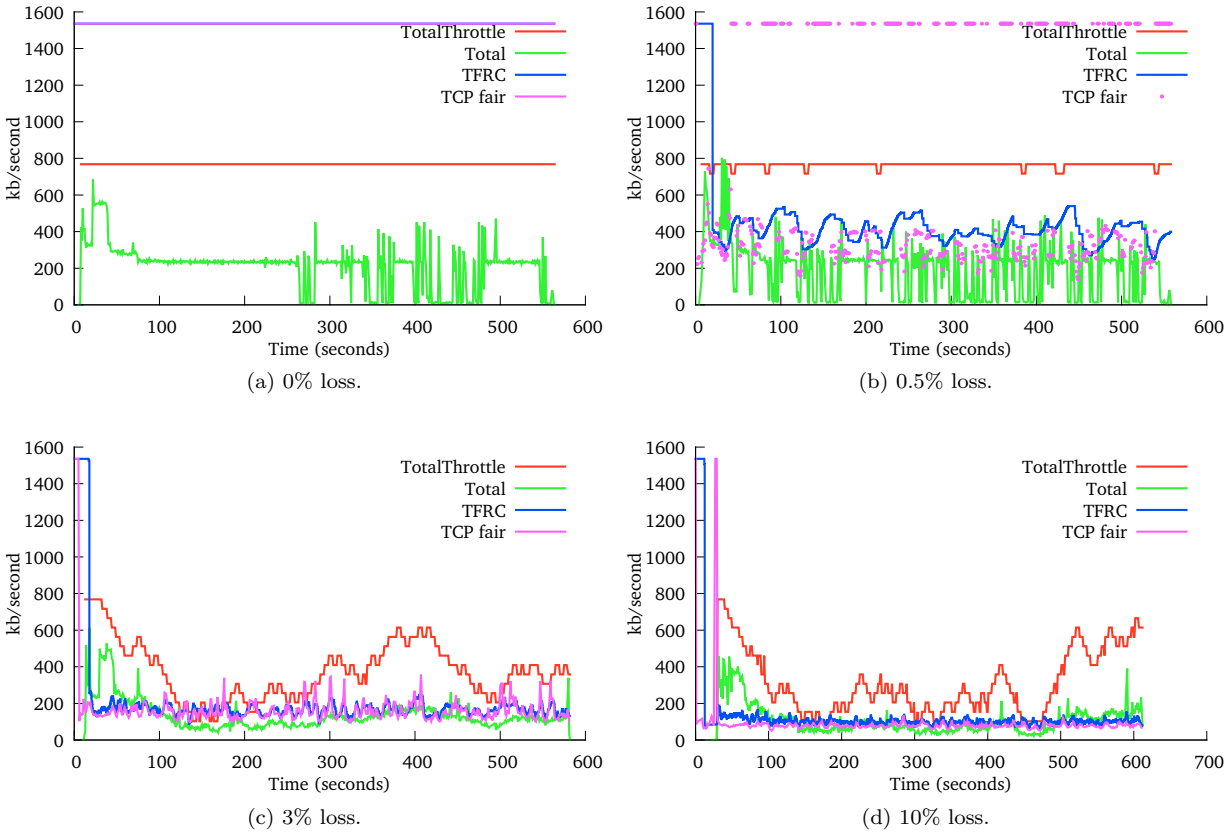
(a) The values for a session where the client is walking around the Sparta island with the throttle value set to 50 Kbps is shown in figure 8.8a. The bandwidth is small, doesn't change very much and is below the throttle value. The mean bandwidth is 44.76 kbps, the median is 39.92 kbps, the maximum is 122.26 kbps, the 25 percentile is 33.80 kbps and the 75 percentile is 52.67 kbps. The variation in the traffic is less for this throttle value than it is for other traces on the same island with the avatar performing the same activity.

(b) The values for a session where the client is walking around the Sparta island with the throttle value set to 1500 Kbps is shown in figure 8.8b. The spikes are larger than when the throttle value is lower. The mean bandwidth is 281.42 kbps, the median is 160.62 kbps, the maximum is 1269.94 kbps, the 25 percentile is 9.94 kbps and the 75 percentile is 499.92 kbps. The throughput is higher than traces with a lower throttle value.

(c) The values for the statistics for different throttle settings of the bandwidth values sampled once per second are shown in figure 8.8c. The 99, 75 and 25 percentiles, the medium, mean and minimum of the bandwidth are shown. These traces are from an avatar walking around the cathedral island. The maximum values are close to the throttle value.

(d) The CDF of the bandwidth for different throttle values for an avatar walking around the cathedral4 island are shown in figure 8.8d. The x-axis is the bandwidth in kbps and the y-axis fraction of the total bandwidth samples that are at or below that rate. The red line shows the values for the throttle value set to 50 kbps, the green line shows the values for 200 kbps, the dark blue line shows the values for 500 kbps, the pink line shows the values for 800 kbps and the light green line shows the values for 1000 kbps. The number of samples at the lowest value increases as the throttle value increases. The peak increases as the throttle value increases. There are vertical regions of the lines that indicates there is throttling of the traffic.

Changes to the throttle value change the behaviour of the system. A higher throttle values there are a larger number of small values and higher peak value. The bandwidth appears to be capped below the throttle value.



(a) 0% loss.

(b) 0.5% loss.

(c) 3% loss.

(d) 10% loss.

Figure 8.9: The bandwidth usage and throttle values for loss levels.

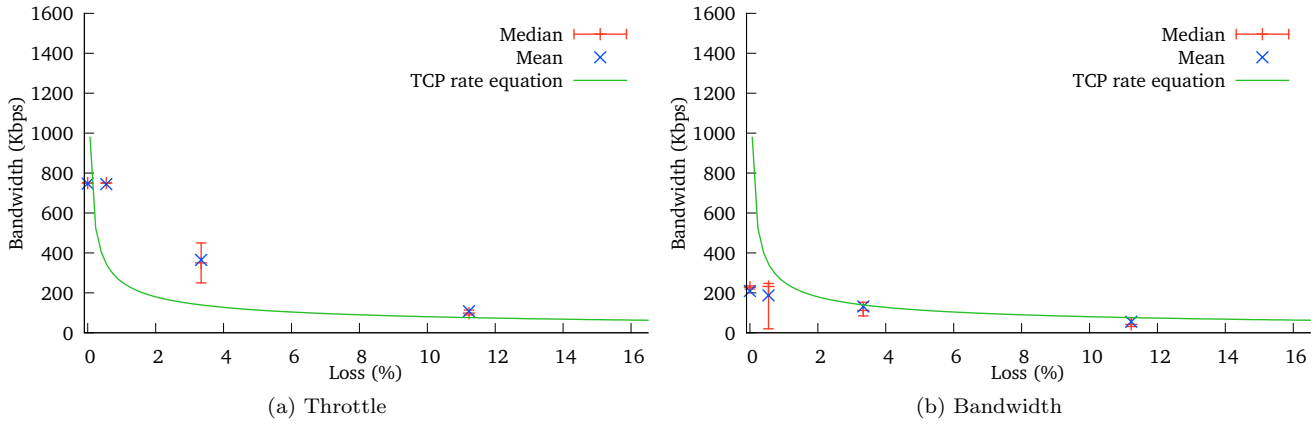
8.3.2 Second Life client and loss levels.

The loss levels were set to different values and the bandwidth utilization for traces are shown in figure 8.9. For these traces the avatar walked around the Sparta island. The x-axis shows the amount of time in seconds since the beginning trace and the y-axis shows the bandwidth in Kbps. The red line shows the throttle value and the green line shows the bandwidth usage. The island used for this is the Sparta island.

(a) The values for a session where the loss rate is set to 0% is shown in figure 8.9a. The throttle value is not changing and the bandwidth is staying below the throttle value. The mean bandwidth is 214 kbps, the median is 234 kbps, the maximum is 684 kbps, the 25 percentile is 224 kbps and the 75 percentile 241 kbps. The loss rate is 0%. The throttle value is smaller than the TCP fair and TCP Friendly Rate Control (TFRC) [72] values. The traffic is not able to reach the throttle value which is itself not as large as the TCP fair value.

(b) The values for a session where the loss rate is set to 0.5% is shown in figure 8.9b. The throttle is reacting a little to the small amount of congestion introduced. The mean bandwidth is 192 kbps, the median is 237 kbps, the maximum is 803 kbps, the 25 percentile is 20.4 kbps and the 75 percentile is 253 kbps. The mean throttle value is 763 kbps, the median is 768 kbps, the maximum is 768 kbps. The mean loss rate is 0.55%. The bandwidth usage value is similar to the TFRC and TCP fair values, these bandwidth values are similar to the bandwidth values for the trace with a loss level of 0%. The bandwidth value is unable to reach the throttle value. The throttle value that the client is calculating is too large.

(c) The values for a session where the loss rate is set to 3% is shown in figure 8.9c. The mean bandwidth is 135 kbps, the median is 115 kbps, the maximum is 613 kbps, the 25 percentile is 86.6 kbps and the 75 percentile is 156 kbps. The mean throttle value is 374 kbps, the median is 358 kbps, the maximum is 768 kbps, the 25 percentile is 256 kbps, the 75 percentile is 461 kbps. The mean loss rate is 3.34% and the median is 3.03%. The bandwidth values are smaller than for 0.5% loss level. The bandwidth value is close to the TCP fair value but does go over the value. The throttle value is larger than the TCP fair value and the utilization is unable to reach it.



(a) Throttle (b) Bandwidth

Figure 8.10: Calculated throttle values and bandwidth usage against loss rate.

(d) The values for a session where the loss rate is set to 10% is shown in figure 8.9d. The throttle adaptation system appears not to be correctly detecting the amount of loss. The mean bandwidth is 228 kbps, the median is 224 kbps, the maximum is 734 kbps, the 25 percentile is 210 kbps and the 75 percentile is 240 kbps. The mean loss rate is 11.7% and the median is 10.9%. The bandwidth values are less close to the TCP fair values than for smaller loss values.

The throttle values and bandwidth usage values against loss rates are shown in figure 8.10. The x-axis shows the percentage of packets that are lost and the y-axis shows the bandwidth in kbps. The red error bars show the upper quartile median and lower quartile. The green line shows the TCP fair equation for a RTT of 150 ms and a packet size of 375 bytes. The blue points are the mean values. The island used for this is the Sparta island, these include the same traces as in figure 8.9

(a) The throttle values against the loss rate is shown in figure 8.10a. The throttle values calculated by the Second Life client are not TCP fair. Low loss levels result in throttle value that is not high enough. At medium to high loss levels the throttle value is to large. When the loss level reaches 10% the calculated throttle value is about right.

(b) The bandwidth against the loss rate is shown in figure 8.10b. The bandwidth usage values are at or below the TCP fair line. There is little difference between the bandwidth used when the loss rate is 0% and 0.5%.

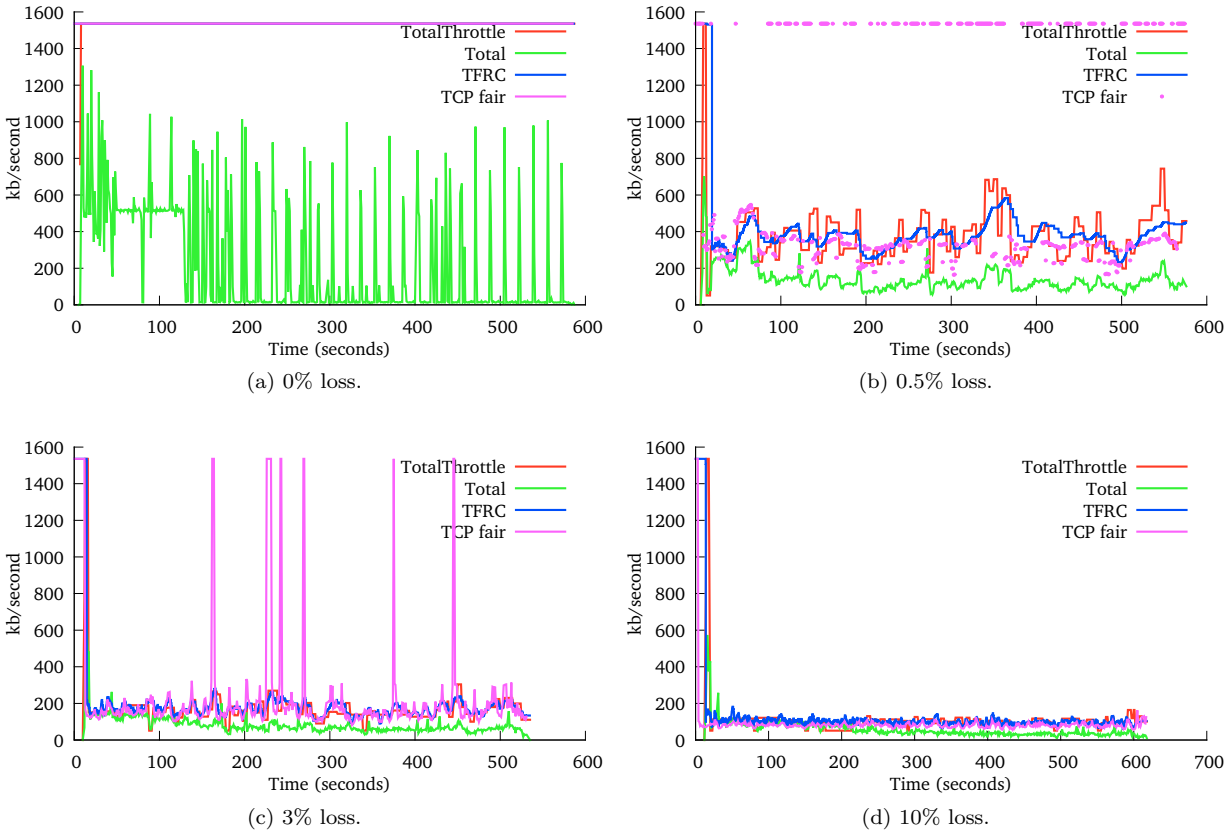
The Second Life client is able to control the amount of bandwidth used by the server, however the client is not able to adapt to congestion well. The throttle levels calculated by the client are higher than the TCP fair values calculated from this trace. The bandwidth usage that is less than the total throttle value. The combination of these things, and the RTT used in these experiments, results in bandwidth usage that is often TCP fair. Under low loss conditions the throttle value does not adjust up to the maximum value and is below the TCP fair rate.

The throttle values are not TCP fair but the bandwidth usage is at high levels of loss. This is caused by the bandwidth values being systematically under the throttle values. At low levels of loss the throttle values are closer to the line and the bandwidth usage is less than the TCP fair value. Next we look at how the Mongoose client behaves with the OpenSim server.

8.3.3 Mongoose client and loss levels.

The Mongoose client was used with the OpenSim server. The loss levels were set to different levels the bandwidth utilization for traces using the Mongoose client are shown in figure 8.11. The x-axis shows the amount of time in seconds since the beginning trace and the y-axis shows the bandwidth in Kbps. The red line shows the throttle value and the green line shows the bandwidth usage. The island used for this is the Sparta island.

(a) The values for a session where the loss rate is set to 0% is shown in figure 8.11a. The Mongoose client is calculating a throttle value greater than the maximum value of 1500 kbps, this value is TCP fair for these conditions. The



(a) 0% loss.

(b) 0.5% loss.

(c) 3% loss.

(d) 10% loss.

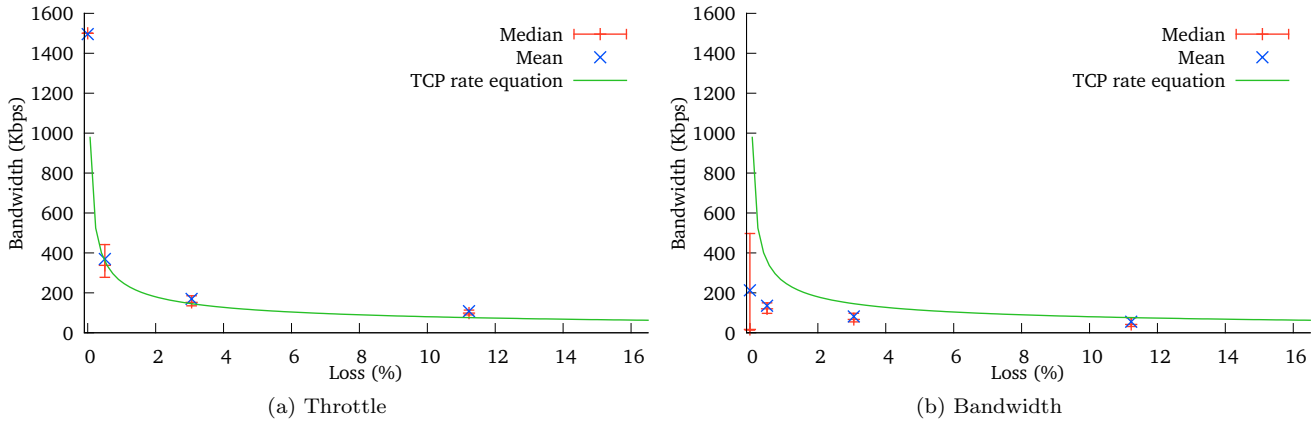
Figure 8.11: The bandwidth usage and throttle values for loss levels.

mean bandwidth is 218 kbps, the median is 17.1 kbps, the maximum is 1306 kbps, the 25 percentile is 13.4 kbps and the 75 percentile is 50.9 kbps. The mean throttle value is 1532 kbps and the median is 1536 kbps, the maximum. The loss rate is 0%. The bandwidth is below the throttle value and appears throttled. The burst in traffic is higher and shorter, this results in the mean being higher and median being lower than in the SLC traces with the default throttle setting.

(b) The values for a session where the loss rate is set to 0.5% is shown in figure 8.11b. The loss is being detected and the client is reacting to it. The mean bandwidth is 139 kbps, the median is 123 kbps, the maximum is 701 kbps, the 25 percentile is 99.5 kbps and the 75 percentile is 154 kbps. The mean throttle value is 378 kbps, the median is 346 kbps, the maximum is 1536 kbps, the 25 percentile is 284 kbps and the 75 percentile is 452 kbps. The mean loss rate is 0.50% and the median loss rate is 0%. The throttle value is close to the TCP fair values when the bandwidth is below the throttle values.

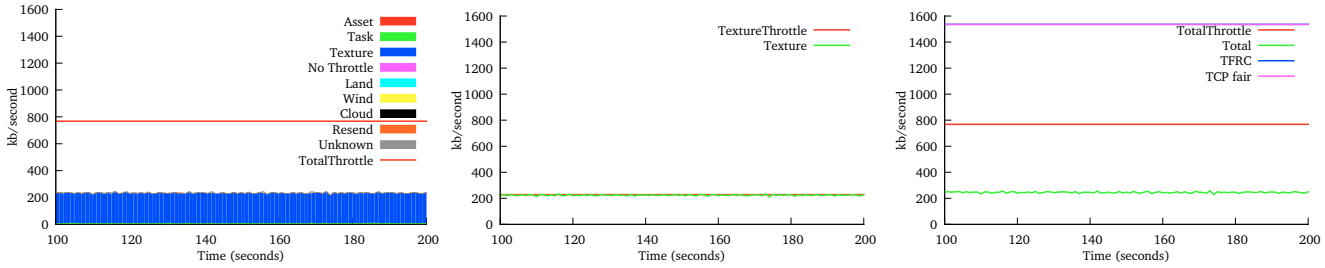
(c) The values for a session where the loss rate is set to 3% is shown in figure 8.11c. The throttle value appears to be controlling the amount of bandwidth used. When the throttle value decreases down to the level of the bandwidth being used the server reacts by decreasing the bandwidth it is using. The mean bandwidth is 83.8 kbps, the median is 68.6 kbps, the maximum is 685 kbps, the 25 percentile is 56.1 kbps and the 75 percentile is 100 kbps. The mean throttle value is 173 kbps, the median is 156 kbps, the maximum is 1536 kbps, the 25 percentile is 138 kbps and the 75 percentile is 190 kbps. The mean loss rate is 3.06% and the median loss rate is 2.60%. The throttle value is at the TCP fair value. The bandwidth is below the throttle value, this is unlikely to be an application limitation as a higher throttle value produces a higher bandwidth across different traces and in this trace.

(d) The values for a session where the loss rate is set to 10% is shown in figure 8.11d. The mean bandwidth is 56.7 kbps, the median is 42.7 kbps, the maximum is 574 kbps, the 25 percentile is 32.1 kbps and the 75 percentile is 76.7 kbps. The mean throttle value is 111 kbps, the median is 101 kbps, the maximum is 1536 kbps, the 25 percentile is 91.9 kbps and the 75 percentile is 117 kbps. The mean loss rate is 11.2% and the median loss rate



(a) Throttle (b) Bandwidth

Figure 8.12: Calculated throttle values and bandwidth usage against loss rate.



(a) The channel bandwidth stacked. (b) Texture bandwidth and throttle. (c) Total bandwidth and throttle

Figure 8.13: The throttle limiting bandwidth usage.

is 10.0%. The throttle values are TCP fair and the bandwidth is below the throttle value.

The throttle values and bandwidth usage values against loss rates are shown in figure 8.12. The x-axis shows the percentage of packets that are lost and the y-axis shows the bandwidth in kbps. The red error bars show the upper quartile median and lower quartile. The green line shows the TCP fair equation for a RTT of 150 ms and a packet size of 375 bytes.

(a) The throttle values against the loss rate is shown in figure 8.12a. The throttle values calculated are by the Mongoose client are about TCP fair. There is most variation in the 0.5% loss value.

(b) The bandwidth against the loss rate is shown in figure 8.12b. The bandwidth usage values are at or below the TCP fair line. The bandwidth values follow a curve which is below the TCP fair value, as the throttle system throttles the bandwidth below the throttle value.

The Mongoose client is calculating a TCP fair throttle value. The bandwidth values are below the throttle value, this confirms the that throttle will not normally be able to use all the bandwidth value. The bandwidth with a loss level of 0% is bursty and not very stable. The bandwidth usage is consistently below the throttle value and increases and decreases as the throttle value increases and decreases. The bandwidth is controlled by the throttle value and the bandwidth is controlled to a value below the throttle value. When the loss level is low the Mongoose client is closer to the TCP fair value than the SLC, at higher loss level Mongoose produce bandwidth that is below the TCP fair value and less close than SLC.

The throttle values are TCP fair but the bandwidth usage is below the TCP fair value. The throttle system limits the individual channels to their values sent from the client and do not share out the unused bandwidth resulting in the usage that is below the TCP fair value.

8.3.4 Per channel throttle values.

The throttle system of OpenSim is limiting the amount of bandwidth to a level below the total throttle value. The traffic consists of packets from different channels that are given a part of the total throttle, these values add up to the total throttle value. The balance in the channel bandwidth does not match the split in bandwidth asked for by the client. It is therefore desirable to look at the traffic split by channel to determine if this situation is what is resulting in the traffic being limited to less than the total throttle value. The traffic of a trace with the avatar walking around the Sparta island is shown split into its component channels in figure 8.13. The x-axis in these figures shows the time in seconds since the beginning of the trace. The y-axis shows the bandwidth in kbps. These time series graphs show the time from 100 seconds to 200 seconds to show the steady state of the system.

(a) The bandwidth used by the different channels is shown in figure 8.13a. The Asset traffic is red, the Task traffic is green, the texture traffic is blue. The line is the total throttle value. The task traffic changes very little throughout these traces and is not at its throttle value. The majority of the traffic is Texture.

(b) The bandwidth used by the texture throttle is shown in figure 8.13b. The red line is the throttle value and the green line is the bandwidth used. The mean bandwidth usage of the texture channel is 181 kbps, the median value is 222 kbps, the maximum is 452 kbps, the 25 percentile is 197 kbps, the 75 percentile is 227 kbps. The texture traffic is not bursty and throttled to use less bandwidth than it would be able to use if it were only limited to the total bandwidth being less than the total allocated bandwidth.

(c) The total bandwidth used is shown in figure 8.13c. The mean bandwidth is 214 kbps, the median is 234 kbps, the maximum is 684 kbps, the 25 percentile is 224 kbps and the 75 percentile 241 kbps. The bandwidth is throttled and the majority of this throttling is the Texture throttle.

The throttle system is limiting the amount of bandwidth more than the total value because it limits each channel individually to its assigned bandwidth. The OpenSim server's throttle system does not seem to share out unused bandwidth between the channels. This results in the server using less bandwidth than the allocated total, it instead limits the amount of bandwidth to the amount allocated to each throttle. If the throttle system was able to share the bandwidth between its throttles it would be able to use more bandwidth without using more than its TCP fair share.

8.3.5 Summary

The OpenSim throttle system is able to effectively limit the amount of bandwidth that it uses. Changes to the throttle value effect the amount of bandwidth that the server is limited to. The throttle adaptation system of the Second Life client does not calculate a TCP fair value. The throttle system limits each channel to the amount that the client sends. This results in the amount of bandwidth that is used by the server being less than the value sent by the client. The combination of these properties results, under the network conditions used, in the server using an amount of bandwidth used being TCP fair for higher low loss levels.

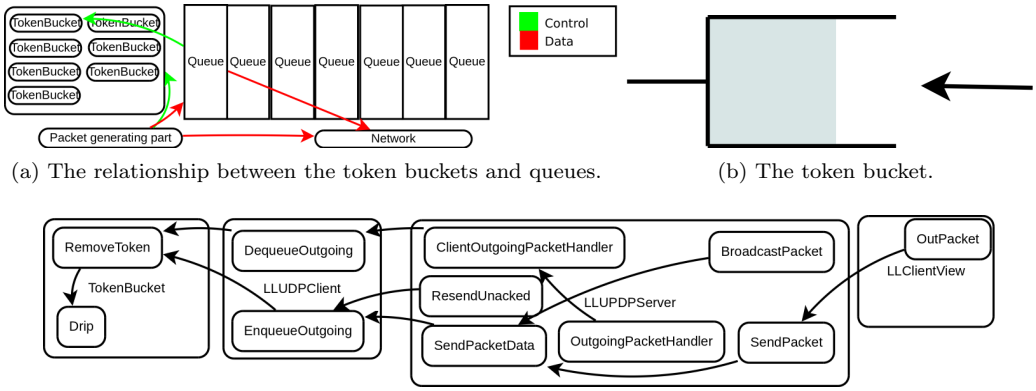
The Mongoose client is able to calculate a TCP fair throttle value and send it to the server. The design of the server side throttle system however results in this combination using less bandwidth than its TCP fair share. The throttle system, under some conditions, is limiting the total bandwidth to the bandwidth to one of the channels. This channel is usually the Texture channel.

The combination of the SLC's throttle value that is too high and the OpenSim throttle system, plus the selected network conditions, results in bandwidth usage that is in the correct range. This correct behaviour happens because of a combination of conditions that are not consistent between all network set-ups. Correcting the problem of the client not sending a TCP fair throttle value by using the Mongoose client, results in bandwidth that is consistently below the TCP fair value.

8.4 OpenSim

OpenSim is an open source virtual world that is compatible with the Second Life client and uses the Second Life Protocol (SLP) [53, 6, 97, 188, 102, 109]. It is written in C# using .Net or mono. The environment it provides is similar to Second Life. It has several possible physics engines, the main one being Open Dynamics Engine [7] (ODE). The ODE physics engine is used in several games. It does not provide the same security model as Second Life.

[7] Russell Smith. Open Dynamics Engine (ODE). `http://www.ode.org/`, 17 October 2009. [Online; accessed 04-April-2011]



(a) The relationship between the token buckets and queues.

(b) The token bucket.

(c) A call graph of the throttle system.

Figure 8.14: The token bucket and queue throttle system.

An OpenSim setup can consist of one program or multiple programs. There are several services that it provides, the user service, the grid service, the inventory service, the asset service, the messaging service and the region service. In earlier versions of OpenSim if it was run as more than one program they had to be provided by a separate program for each service. In the version used the grid, inventory and asset services can be run as a single program. When OpenSim is run as several programs it is possible to have several region services running on different machines.

The user server handles the authentication of users the information about avatars. The grid server holds information about the location of islands in the grid. The inventory holds information about the user's inventory with references to the assets. The asset server contains the assets.

8.4.1 OpenSim Packet throttle system.

The throttle system maintains compatibility with the Second Life client by using a compatible channel grouping. The channels are kept the same except the Task throttle is split into State, which contains the state of non avatar object and Task which contains all of the other packets that Second Life puts in Task. This takes the majority of the bytes of Task out of that channel leaving mostly avatar traffic. This achieves most of what the alternative channel design, described in section 8.1, was intended to achieve. In that is isolates the avatar traffic from other traffic and protects its bandwidth.

There are separate `TokenBuckets` [26] for each channel in each circuit, these are used to limit the amount of bandwidth used. The token buckets are emptied as packets for its channel are sent out onto the network. The token buckets work by filling at a constant rate up to the burst size value. When the bytes are to be sent out onto the network the number of tokens corresponding to the bytes sent are removed from the bucket, if there are not sufficient then the packet is queued. The `LLUDPClient` class handles the circuits and holds queues for the different channels. The `EnqueueOutgoing` function finds the throttle and queue. It tries to remove the number of tokens corresponding to the packet size, if there were enough tokens the function returns false otherwise the packet is queued and the function returns true. This function is called from two other functions which send out the packet if the enqueue function returns false.

The function called to remove the token from the bucket calls the `Drip` function which refills the bucket by the correct amount for the length of time since the last call to `Drip` up to the maximum burst size, which is the amount of traffic that can be sent out in one second. This happens before the attempt is made to remove the number of tokens equal to the amount of data to be sent. When there are packets being constantly sent out the burst rate limits the length of bursts in which queued packets will be sent and also the maximum amount that can be sent after a delay when no packets are being sent out. After the number of tokens has been checked the bucket checks if there is a parent bucket and then calls remove tokens on it if it exists. If there is no parent bucket or the call to remove tokens returns true, the number of tokens is reduced by the number of tokens to be removed.

The `DequeueOutgoing` function loops through channels and tries to send a packet from queue. This function is periodically called by `ClientOutgoingPacketHandler` which is called by `OutgoingPacketHandler` for each circuit. This function runs a loop that sleeps for the clock granularity and then tries to send any queued packets. The function performs the sending of re-sent packets and ACK packets as well. If the packets were sent the function doesn't sleep

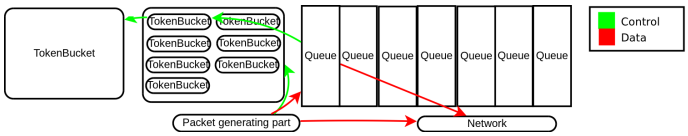


Figure 8.15: The relationship between the token buckets and queues in OpenSim TR.

so the function caries on trying to send until it can't any more and then sleeps.

Packets are either sent or added to the queue by `SendPacketData` and `ResendUnacked`. `ResendUnacked` is called by `ClientOutgoingPacketHandler` to resend reliable packets that haven't been acknowledged and have timed out. The `SendPacketData` function is called by two functions `SendPacket` and `BroadcastPacket`, `BroadcastPacket` sends the packet to all the clients in the scene. `SendPacket` is called by the functions that send ACKs and Ping packets, it is also called by the `OutPacket` functions in `LLClientView`. The `OutPacket` functions are called by the functions in `LLClientView` that send the different types of packet. The throttle type of the packet is provided by these functions. This is illustrated by the diagram in figure 8.14.

This throttle system limits the sending rate to the correct value when there are always packets to send. If there are not packets to send for a second and then a large number of packets to send, the entire burst rate of packets will be sent out and the rest will be queued. The packets will then be sent out, this will result in almost two seconds worth of data being sent in one second. The long term average will not exceed the set value but the value for one second can be up to twice the set value.

8.4.2 Modified OpenSim server

The problem with the combination of the Mongoose client and the OpenSim server is that it limits each channel to the amount of bandwidth that the client allocates to it and the Mongoose client calculates a TCP fair total throttle value and shares this amongst the channels in a predefined way. It is desirable that the server limit its traffic to its TCP fair share and that it be able to use all of its TCP fair share when it has sufficient data to send. As the server uses the share that the client send to it, it would be possible to modify Mongoose so that it detects the OpenSim server and sends different throttle values.

If a modified client were used it would have to change the share of the bandwidth that each channel is given. This would mean that the Texture channel would be given more bandwidth. The other channels are however still used so, the options are to remove bandwidth from other channels to give to the Texture channel in which case they would be more limited in their bandwidth usage than intended. The alternative is to increase the Texture channel without changing the others this would result in the server sometimes using more than its TCP fair share of the bandwidth.

The alternative to modifying the Mongoose client further is to modify the OpenSim server so that it can share out unused bandwidth to the channel that have more data to send. This allows the total value to be sent from the server to be limited to the TCP fair value sent from the client and to uses as much of that bandwidth as it has data to use.

There are different ways in which the OpenSim server could have been modified. The token buckets could have been replaced with another type of traffic limiting system, this would require more changes to the server than modifying the token bucket system. The token bucket system works well to limit the bandwidth used so it is reasonable to keep it. It would be possible to limit the bandwidth for each channel to there allocated bandwidth when the bandwidth usage reaches the maximum or to change the bandwidth share to allow the channels with more bandwidth to use the unused bandwidth.

The design of the throttle system is a token bucket for each channel and parent bucket for all of those buckets that limits the overall bandwidth for the circuit. The overall bucket does not limit the bandwidth in the unmodified OpenSim server unless it is configured to. The OpenSim server's throttle system is good at limiting to what the client requests, if the per channel throttles are to be enforced rather than the total throttle value. This is not the behaviour that Second Life has and it is not important to the network that the share between OpenSim's channels is maintained. The behaviour of guaranteeing that a channel can get the amount of bandwidth that is allocated to it is important and stopping the total bandwidth usage exceeding the total throttle value is important. The behaviour of not using the spare bandwidth that channels are not using is not desirable as it reduces the utility the system gets from the network and reduces the bandwidth usage below the correct value.

It is therefore desirable to modify the throttle system so that it limits the total bandwidth usage to the total throttle value. It should also not limit the channels to their throttle value unless this will result in another channel

not being able to use up to its throttle value. The design of the new throttle system is such that when packets are to be sent and there is sufficient throttle left they are sent out and otherwise they are queued. This results in a two layer throttle system. The queued packets are then sent out through a round robin system that goes through the channels and sends out a packet from each channel that has one to send each time round. The design of the modified system will not queue packets until sending is stopped by the total throttle and then the sending from the queues is limited by the per channel throttles.

The design of the new throttle system is that there is a top level throttle system to limit the total bandwidth and channel level throttle to distribute the bandwidth between the channels in the desired proportions, this is illustrated by figure 8.15. At both levels of this system token buckets are used where the tokens fill up at a constant rate and there is a burst rate value that limits that number of tokens that can accumulate. The token buckets at both levels are the same but the channel level ones do not limit transmission so long as there is unused bandwidth. There is limitation by the channel token buckets when the there is not spare bandwidth, the token buckets then distribute the bandwidth between channels. When the bandwidth runs out the packets are queued and the system iterates through the queues and token buckets. The system moves through the channels in a round robin system until no packets can be sent.

The throttle system has been modified so that the total bucket limited the bandwidth usage to the maximum value. The channel buckets were modified to not limit the amount of bandwidth usage of packets that haven't been queued. This was achieved by setting the total buckets burst rate to, initially, the total bandwidth and creating a new function in `TokenBuckets`. The new function calls the parent bucket's remove tokens function if there is one and the bucket doesn't have sufficient tokens and returns true if this function returns true.

This should mean that when there is data to send the bandwidth reaches the global throttle value and that each channel gets at least its throttle value of bandwidth sent. So traffic from one channel should not be able to interfere with the traffic from other channels. The design of the OpenSim channels is that the non-avatar objects are separated from the Task channel they should not interfere with avatar traffic.

With a burst rate of one second of data the modified server is able to send at the rate sent from the client. The rate however spikes up to twice the rate for the reason described in section 8.4.1. The server was further modified to use a burst rate that is a fraction of a second's worth of data. When the value is too small the server is unable to achieve the specified sending rate, though the sending rate that it does achieve is more stable. This modified version of OpenSim will be called OpenSim TR (Throttle Redistributing).

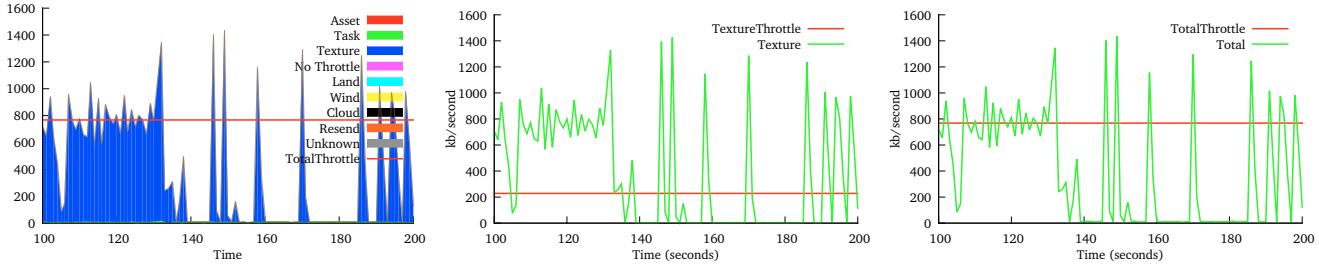
8.4.3 Summary

The OpenSim server provides environment that is similar to that provided by Second Life. It uses the same protocol and responds to the throttle packets sent from the client. The throttle system used by OpenSim is different from the one used by Second Life and it splits up the task channel packets into two channels, Task and State. This throttle system is more strict than the one used by Second Life. The modifications made to the throttle system maintain the strictness of the overall throttle but remove the strictness of the per channel throttles and allows unused throttle to be redistributed.

8.5 Measurement of Modified OpenSim server.

The aim of this section is to evaluate the modified version of OpenSim described in the previous section and look at the parameter choices made. The experiments with the OpenSim TR server were performed using the same methodology as the unmodified OpenSim server. The traffic was directed over a test network that allowed the loss and RTT to be controlled. The traffic was captured at both ends of the network, this is described in section 8.2.1. A third server was used with a copy of the database and settings from the OpenSim server, that holds the Sparta, with the modified OpenSim server programs. The Sparta island was used because it was used in the previous section and a new server was used to allow experimentation to be done with these islands without disrupting access the existing server. The database was dumped from the OpenSim server and loaded on to a separate machine. The settings files were copied and the modified OpenSim server was installed and run. This sections will consist of the following:

- A test of the ability of OpenSim TR to redistribute bandwidth to the channels.
- An examination of the effect of the size of the burst size of the total throttle token bucket.



(a) The channel bandwidth stacked. (b) Texture bandwidth and throttle. (c) Total bandwidth and throttle

Figure 8.16: The throttle limiting bandwidth usage.

- The effect of loss rates on the throttle value and bandwidth usage.

8.5.1 Per channel throttle values.

This section investigates the ability of OpenSim TR to redistribute bandwidth to channels that have more data to send. The result of the modified client is shown in figure 8.16. The x-axis in these figures shows the time in seconds since the beginning of the trace. The y-axis shows the bandwidth in kbps. The three graphs show the bandwidth in different ways. The first shows the different channels staked, the second shows just the texture channel and the third shows the total with the throttle. These time series graphs show the time from 100 seconds to 200 seconds to show the steady state of the system.

(a) The bandwidth used by the different channels is shown in figure 8.16a. The Asset traffic is red, the Task traffic is green, the texture traffic is blue. The line is the total throttle value. The majority of the data belongs to the Texture channel.

(b) The bandwidth used by the texture throttle is shown in figure 8.16b. The red line is the throttle value and the green line is the bandwidth used. The mean bandwidth usage of the texture channel is 187 kbps, the median value is 0 kbps and the maximum is 6573 kbps. The texture traffic is bursty and spikes above the total throttle value. The texture channel is make use of bandwidth from other channels, this reduces the time for downloading texture making the system more responsive.

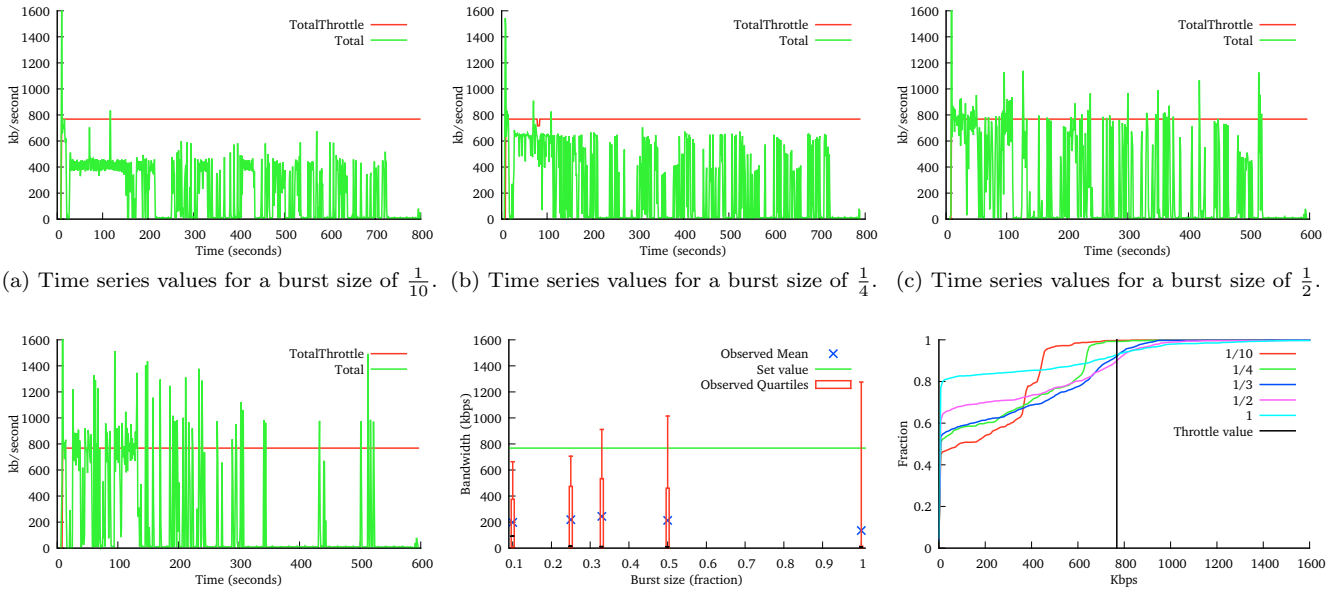
(c) The total bandwidth used is shown in figure 8.16c. The mean bandwidth is 205 kbps, the median is 10.2 kbps, the maximum is 2270 kbps, the 25 percentile is 8.87 kbps and the 75 percentile is 207 kbps. The bandwidth reaches the throttle value and oscillates around it for part of the trace. The bursts are above the throttle values. This graph shows that the bandwidth is consistent with the total throttle. As the global throttle is controlling the bandwidth usage the burst size of this throttle is important and allows bursts over the throttle value.

The throttle system is limiting the amount of bandwidth more than the total value because it limits each channel individually to it assigned bandwidth. If the throttle system were able to share the bandwidth between its throttles it would be able to use more bandwidth without using more than its TCP fair share. The bandwidth is more bursty than the unmodified OpenSim, this means that the textures are downloaded faster providing better service to the user. The unmodified OpenSim server was not able to use the unused bandwidth. This new version is able reuse bandwidth from other channels. This reduces the time taken to download data from the server to the client. The bandwidth is limited by the global throttle.

8.5.2 Total throttle burst size.

The server was modified to change the burst size of the overall throttle bucket. Versions of the server with burst size of different fractions of the throttle value were tested, the results are shown in figure 8.17. The fractions that were used are $\frac{1}{10}$, $\frac{1}{4}$, $\frac{1}{3}$, $\frac{1}{2}$ and 1. The Sparta island was used for this experiment and the avatar walked around the island.

(a) The bandwidth and throttle values against time for a trace where the burst size for the overall throttle is set to amount of bandwidth to be sent in $\frac{1}{10}$ of a second, is shown in figure 8.17a. The mean bandwidth value is 202



(a) Time series values for a burst size of $\frac{1}{10}$. (b) Time series values for a burst size of $\frac{1}{4}$. (c) Time series values for a burst size of $\frac{1}{2}$.

(d) Time series values for a burst size of 1. (e) Box plot of bandwidth against burst size. (f) Cumulative distribution frequency

Figure 8.17: Bandwidth usage against burst size as a fraction of the throttle value.

kbps, the median is 93.7 kbps, the maximum is 1688 kbps, the 25 percentile is 9.38 kbps, the 75 percentile is 384 kbps. The bandwidth only spikes over the throttle value once after it has been set, the majority of the time the bandwidth is throttled to a level that is not at the throttle value.

(b) The bandwidth and throttle values against time for a trace where the burst size for the overall throttle is set to the amount of bandwidth to be sent in $\frac{1}{4}$ of a second, is shown in figure 8.17b. The mean bandwidth value is 224 kbps, the median is 14.9 kbps, the maximum is 1544 kbps, the 25 percentile is 9.23 kbps, the 75 percentile is 486 kbps. The bandwidth level that this version of the server is able to reach is closer to the throttle value that for $\frac{1}{10}$.

(c) The bandwidth and throttle values against time for a trace where the burst size for the overall throttle is set to amount of bandwidth to be sent in $\frac{1}{2}$ of a second, is shown in figure 8.17c. The mean bandwidth value is 251 kbps, the median is 11.3 kbps, the maximum is 2794 kbps, the 25 percentile is 9.02 kbps, the 75 percentile is 556 kbps. The bandwidth values are closer to the throttle value than the others.

(d) The bandwidth and throttle values against time for a trace where the burst size for the overall throttle is set to amount of bandwidth to be sent in 1 of a second, is shown in figure 8.17d. The mean bandwidth is 205 kbps, the median is 10.2 kbps, the maximum is 2270 kbps, the 25 percentile is 8.87 kbps and the 75 percentile is 207 kbps. The bandwidth is able to reach the throttle value and it spikes over the throttle value more of the time than the lower burst values.

(e) The mean, median, 99 percentile, minimum and up and lower quartiles for the per second bandwidth are shown in figure 8.17e. The x-axis is the burst size as a fraction of the per second bandwidth value. The y-axis is the bandwidth in kbps. The 99 percentile bandwidth values are larger than the throttle value.

(f) The Cumulative distribution frequency of the bandwidth values is shown in figure 8.17f. The smaller burst sizes have flat areas of their bandwidth usage below the throttle value. This effect is less for the larger burst sizes. The tenth line has 45.0% of the samples at or below 8 kbps. The quarter line has 50.9% of the samples at or below 8 kbps. The third line has 53.5% of the samples at or below 8 kbps. The half line has 61.2% of the samples at or below 8 kbps. The one line has 77.1% of the samples at 8 kbps.

The amount of bandwidth used by the server dependence on the size of the burst value. The bandwidth value spikes over the throttle value when burst size is large and when the burst size is small the bandwidth does not reach the throttle value. The traffic for the $\frac{1}{10}$ is smoother and more throttled and less application limited this leads to slower downloads. The value of $\frac{1}{2}$ seems to be good balance between these two extremes.

8.5.3 Mongoose client and loss levels with the modified client.

This section looks at the effect of loss levels on the throttle values that the Mongoose client calculates and the effect this has on the modified OpenSim server's bandwidth usage. This will make is possible to evaluate if the new server works correctly. The loss levels were set to different values. The bandwidth utilization for traces using the Mongoose client and the modified OpenSim server are shown in figure 8.18. The x-axis shows the amount of time in seconds since the beginning trace and the y-axis shows the bandwidth in Kbps. The red line shows the throttle value and the green line shows the bandwidth usage. In the distribution frequency graphs y scales are different, so care must be taken in comparing their values.

(a) The values for a session where the loss rate is set to 0% is shown in figure 8.18a. The Mongoose client is working as expected, calculating a throttle value which is limited to the maximum value of 1500 kbps. The mean bandwidth is 145 kbps, the median is 10.5 kbps, the maximum is 5,840 kbps, the 25 percentile is 9.04 kbps and the 75 percentile is 25.0 kbps. The mean throttle value is 1412 kbps and the median is 1536 kbps, the maximum. The loss rate is 0%. The server is able to use the full bandwidth, consequently the textures are more quickly downloaded improving the interactivity.

(b) The values for a session where the loss rate is set to 0.5% is shown in figure 8.18b. The loss is being detected and the client is reacting to it. The mean bandwidth is 154 kbps, the median is 52.1 kbps, the maximum is 6771 kbps, the 25 percentile is 9.93 kbps and the 75 percentile is 249 kbps. The mean throttle value is 375 kbps, the median is 337 kbps, the maximum is 1536 kbps, the 25 percentile is 263 kbps and the 75 percentile is 433 kbps. The mean loss rate is 0.66% and the median loss rate is 0%. This shows the throttle adjusting at low loss level and the bandwidth adjusting to this.

(c) The values for a session where the loss rate is set to 3% is shown in figure 8.18c. The throttle value is controlling the amount of bandwidth used. When the throttle value decreases down to the level of the bandwidth being used the server reacts by decreasing the bandwidth it is using. The mean bandwidth is 138 kbps, the median is 125 kbps, the maximum is 3961 kbps, the 25 percentile is 58.3 kbps and the 75 percentile is 160 kbps. The mean throttle value is 197 kbps, the median is 169 kbps, the maximum is 1536 kbps, the 25 percentile is 144 kbps and the 75 percentile is 191 kbps. The mean loss rate is 3.46% and the median loss rate is 2.63%. This shows the throttle adapting to loss at a higher level of loss.

(d) The values for a session where the loss rate is set to 10% is shown in figure 8.18d. The mean bandwidth is 81.7 kbps, the median is 43.0 kbps, the maximum is 4595 kbps, the 25 percentile is 26.3 kbps and the 75 percentile is 71.7 kbps. The mean throttle value is 101 kbps, the median is 104 kbps, the maximum is 1536 kbps, the 25 percentile is 87.0 kbps and the 75 percentile is 107 kbps. The mean loss rate is 15.6% and the median loss rate is 10.5%.

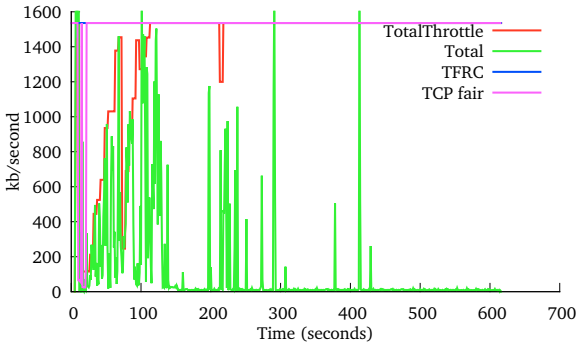
(e) The distribution frequency of per second bandwidth usage for a connection where the avatar was walking around the Sparta island with a loss level of 0% is show in figure 8.18e. The x-axis is the bandwidth in kbps and the y-axis shows the fraction of the time that the bandwidth is at that level. There is one peak at 8 kbps, which is 67.1%. The bandwidth spikes to high levels but spends most of its time at a very low level.

(f) The distribution frequency of the bandwidth with a loss rate of 0.5% is shown in figure 8.18f. The axes have the same meaning as in the previous figure. There is one peak at 8 kbps, which is 37.7%. The bandwidth spikes to high levels but spends most of its time at a low level.

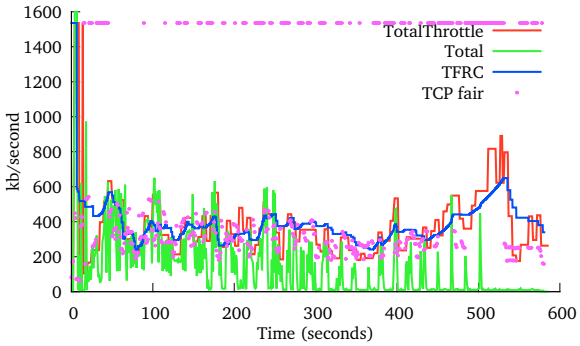
(g) The distribution frequency of the bandwidth with a loss rate of 3% is shown in figure 8.18g. The axes have the same meaning as in the previous figure. There are two peaks of similar size, at 57 kbps and 144 kbps. The smaller peak is at 296 kbps.

(h) The distribution frequency of the bandwidth with a loss rate of 10% is shown in figure 8.18h. The axes have the same meaning as in the previous figure. There is one large peak at 24 kbps and there is smaller peak at 64 kbps.

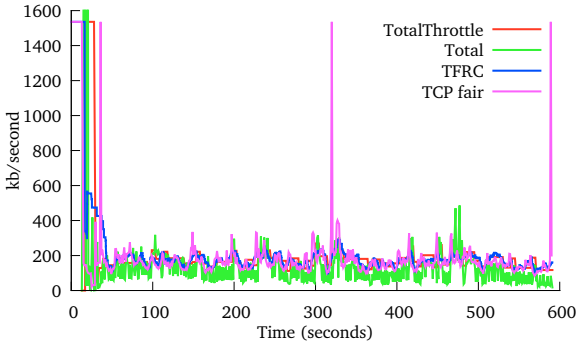
The Mongoose client is calculating a TCP fair throttle value. The bandwidth values are closer to the throttle values with this modified OpenSim server. The OpenSim TR server with the Mongoose client works as desired. At low loss levels the textures are downloaded quickly and at high loss levels the bandwidth usage is stable and TCP fair.



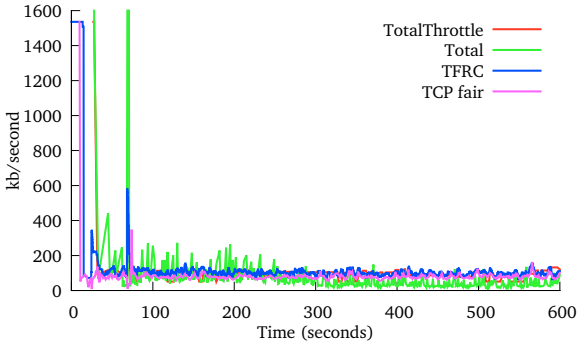
(a) 0% loss.



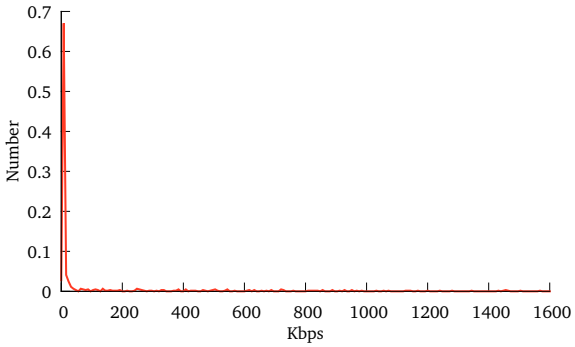
(b) 0.5% loss.



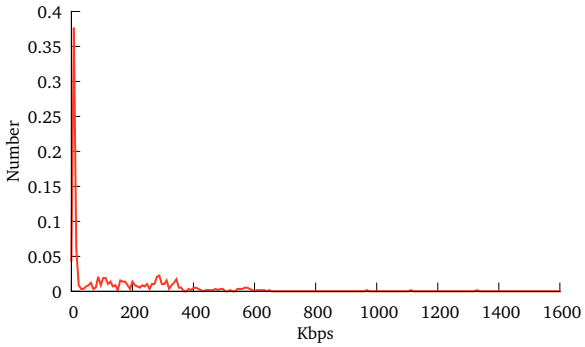
(c) 3% loss.



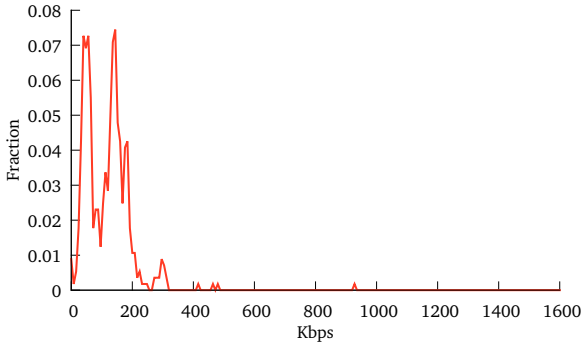
(d) 10% loss.



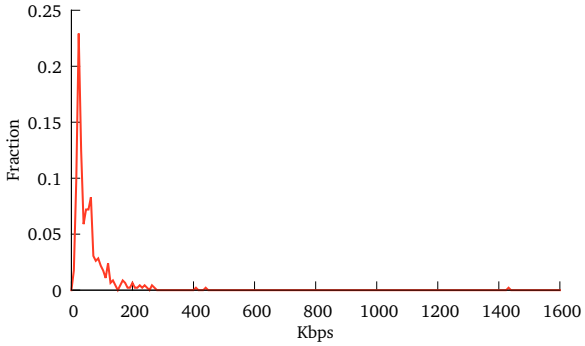
(e) Bandwidth distribution frequency for 0%.



(f) Bandwidth distribution frequency for 0.5%.

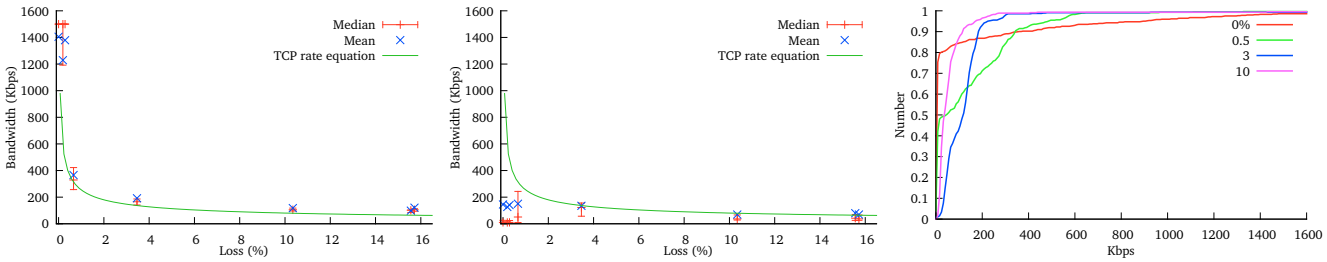


(g) Bandwidth distribution frequency for 3%.



(h) Bandwidth distribution frequency for 10%.

Figure 8.18: The bandwidth usage and throttle values for loss levels.



(a) Throttle (b) Bandwidth (c) Bandwidth

Figure 8.19: Calculated throttle values and bandwidth usage against loss rate.

8.5.4 Throttle and bandwidth against loss rate.

The throttle values and bandwidth usage values against loss rates are shown in figure 8.19. The x-axis shows the percentage of packets that are lost and the y-axis shows the bandwidth in kbps. The red error bars show the upper quartile, median and lower quartile. The green line shows the TCP fair equation for a RTT of 150 ms and a packet size of 375 bytes. The blue points are the mean values.

(a) The throttle values against the loss rate is shown in figure 8.19a. The throttle values close to the TCP fair line and fit into a shape that is the same as the TCP fair values, this indicates that the Mongoose client is working as intended. There is more variation in the throttle values when the loss rate is lower.

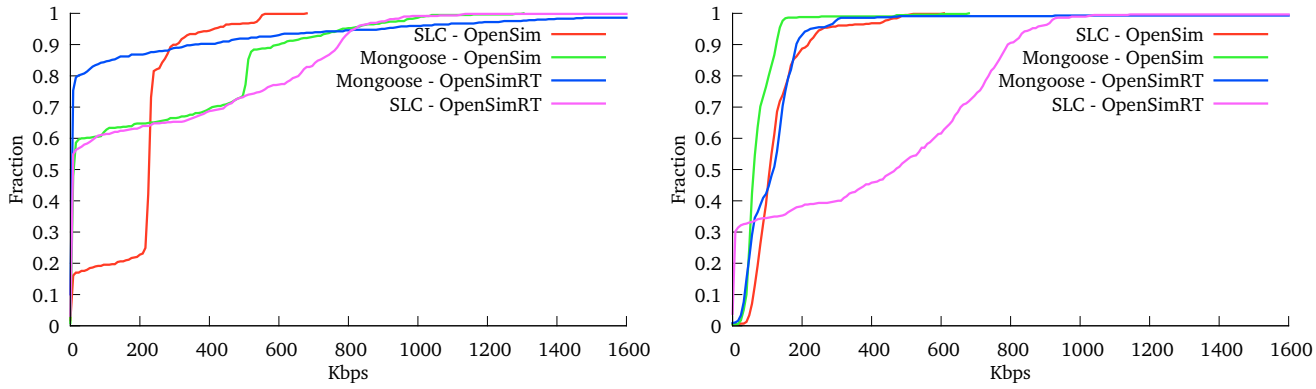
(b) The bandwidth against the loss rate is shown in figure 8.19b. The mean bandwidth usage does not change significantly between 3% loss and 0% loss, the median bandwidth is however much smaller. The low loss level traces have lower median bandwidth and higher burst in traffic. The OpenSim traffic is network and we hypothesize application limited, in the case of low loss levels the application is the limiting factor on bandwidth usage for the majority of the samples. The shorter bursts mean that when the avatar comes close to an object or texture it is more quickly transferred to the client than at higher loss levels. The small difference between the mean, median and quartiles at higher loss levels means that the traffic is less bursty and it limited by the throttle system rather than the application. The usage at medium to high loss levels is TCP fair. The usage at low loss levels is below the TCP fair value as it is application limited.

(c) The cumulative distribution frequency of the bandwidth for different loss levels is shown in figure 8.19c. The x-axis shows the bandwidth in kbps and the y-axis shows the fraction of the total samples that are of that bandwidth or less. The red line shows the values for 0% loss, the green line shows the values for 0.5% loss, blue line shows the values for 3% loss and the pink line shows the values for 10% loss. The 0% loss the majority of the bandwidth samples are 8 kbps. At low loss levels there are more low samples and more high samples, the bandwidth spends most of its time at low levels and increases for a short period of time when things are quickly downloaded. The application is able to use bandwidth when it has use for it. At medium and high loss levels there are more samples in the middle so they are more throttled and the bandwidth is more stable.

The throttle calculation system is able to correctly calculate a value that is close to the TCP fair values. The bandwidth is effected by the throttle values sent from the client to the server. The throttle values calculated are TCP fair. At the 0% and 0.5% loss levels the majority of the bandwidth levels are low, but there are short bursts that are larger. This gives a mean bandwidth that is larger than the median and closer to the throttle value. The traffic produced the OpenSim TR server and Mongoose client is bursty at low loss levels, which would indicate that the network not heavily loaded, when the network can handle this type of traffic. The traffic produced when the loss level is high, which would indicate that the network is under heavy load, is not bursty so does not put undue stress on the network. The combination of the Mongoose client and OpenSim TR server does the right thing at low, medium and high loss levels.

8.5.5 Summary

The modified system works as expected. The version of the system first created where the total bucket is the same as the others but with a drip and burst size equal to the per second total throttle value has a mean bandwidth which



(a) For 0% loss.

(b) For 3% loss.

Figure 8.20: CDF of bandwidth for different combinations of the clients and servers.

is correct but the per second bandwidth can be twice as large. This is because the design of the buckets allows two seconds traffic to happen in one second so long as the previous second had no traffic. Spikes in the bandwidth this large are not desirable so versions of the server where the burst size was different fractions of the one second bandwidth were created. Smaller burst sizes produce a rate that is smaller than the throttle value and a value of one produce burst that are too large. The value of $\frac{1}{2}$ was concluded to be provide best balance between these two problems.

The OpenSim TR server is able to produce a TCP fair sending rate when given a TCP throttle value by the Mongoose client. The OpenSim TR server is able to share out the unused bandwidth to channels that have more data to send. The size of the burst size value effects the how well the server limits the traffic to not exceeding the throttle value, a burst size of half a second's traffic provides the best balance between control and being able to use the correct amount of bandwidth. The Mongoose client is able to adapt to different loss levels and the server is able to control the bandwidth. The throttle values against loss are in the correct shape and the shape of the probabilistic distribution frequency of bandwidth is in the correct shape.

8.6 Comparison between different versions.

The cumulative distribution frequency of the bandwidth for the different combinations of the servers and client is shown in figure 8.20. The clients are the standard Second Life client and the Mongoose client which is described in chapters 6 and 7. The servers are the unmodified OpenSim server and the OpenSim TR server described in section 8.4.2. The x-axis shows the bandwidth in kbps and the y-axis shows the fraction of the total samples that are of that bandwidth or less. The red line shows the values for the unmodified Second Life client with the unmodified OpenSim server. The green line shows the Mongoose client with the unmodified OpenSim server. The blue line shows the Mongoose client with the OpenSim TR server. The pink line shows the unmodified Second Life client with the OpenSim TR server.

(a) The CDF for the traces with a loss level of 0% are show in figure 8.20a. The traces with the unmodified OpenSim server have a large number of samples at about the same level, this is the level that the bandwidth is limited to. The Mongoose client with the modified OpenSim server has a throttle level that is high enough that it doesn't limit that amount of bandwidth used. The Second Life client with OpenSim TR has a change in the steepness of the line at the point were the throttle value is.

(b) The CDF for the traces with a loss level of 3% are shown in figure 8.20b. The throttle values sent by the unmodified Second Life client are larger than the those generated by the Mongoose client. The combination of this larger throttle value and the throttle system of OpenSim results in a similar behaviour to the Mongoose client with the OpenSim TR server. The Mongoose client with the unmodified OpenSim server is throttled to a lower level, that is below the TCP fair level. The Second Life client, with its non-TCP fair throttle values, with the OpenSim TR server results in bandwidth usage is above the TCP fair level.

The Second Life client with the unmodified OpenSim server under these network conditions results in a bandwidth usage that is TCP fair for several levels of loss. At low levels of the loss because the client does not adapt to the

maximum value the bandwidth is less than it should be as it is throttled below the TCP fair value. The client asks for too much bandwidth and the server doesn't give it the amount of bandwidth that it asks for. This combination results in bandwidth that is TCP fair for medium to high loss levels. This is shown in figure 8.10.

The Mongoose client with the unmodified OpenSim server works more consistently though it produces a bandwidth level that is lower than the TCP fair level. At low loss levels the bandwidth usage is larger than for SLC. The throttled level is higher because the throttle value is larger, this results in the bandwidth being low for more of the time. This is good as it means that when object and textures are encountered they are more quickly downloaded. At higher loss levels the bandwidth is lower because the throttle value is lower and the server side mechanism is limiting the bandwidth to below the throttle value. The Mongoose client is asking for a TCP fair amount of bandwidth but the server is not giving it the amount of bandwidth it asks for. This combination works for low loss levels but less well for high loss levels. This is shown in figure 8.12.

The Second Life client with the OpenSim TR server produces bandwidth usage that is not TCP fair and is similar to the behaviour of the Second Life servers. The Second Life client's throttle calculation does not take the RTT into account when calculating its throttle value so for a shorter RTT but the same loss level it would send the same throttle value, resulting in bandwidth usage that is not TCP fair. This combination is better at low loss levels because the server is throttling to a higher level than the unmodified server. At higher levels of loss this combination allows the bandwidth to be larger than the TCP fair value. This results in the traffic being bursty as it is application limited. Bursty traffic should be avoided when there is high load on the network as indicated by the high level of loss. Bursty traffic is not a problem when the network is not under load which is indicated by low levels of loss. The SLC is asking for more than its TCP fair share of the bandwidth and getting what it asks for.

The OpenSim TR server with the Mongoose client brings the bandwidth usage up closer to the TCP fair level. At low levels of loss the bandwidth is mostly at a low level for the majority of the time. This means that the bandwidth is more bursty than the other combinations, this means that the requests for objects and texture are more satisfied quickly. At higher levels of loss the client calculates and sends a throttle value that is TCP fair. The server uses this value to throttle the bandwidth to a TCP fair level. This combination the Mongoose client and the OpenSim TR server unlike the other combinations is able to produce TCP fair bandwidth usage at low, medium and high loss levels. The traffic is bursty when there is low loss and so the network is unloaded and is stable, smooth and TCP fair at high loss levels. This is shown in figure 8.19.

The SLC calculates a throttle value that is not TCP fair. The Mongoose client calculates a throttle value that is TCP fair. The unmodified OpenSim server throttle system limits its bandwidth to less than the throttle value that client sends to it. The OpenSim TR server's throttle system allows it to use the amount of bandwidth that the client requests. The combination of the Mongoose client and the OpenSim TR acts correctly at low medium and high loss levels.

8.7 Conclusion

The properties of the Second Life throttle system have been investigated in chapter 7. The OpenSim server is a different server not based on source code of the Second Life servers so it has a different throttle system. The functioning of this system has been investigated in detail in part facilitated by the availability of source code.

The OpenSim server produces traffic which is similar to the Second Life servers, using the same protocol. The traffic is different, it uses fewer types of packet and specifically less packets that are marked reliable. The throttle value calculated by the standard Second Life client is, like with Second Life server, not TCP fair. For the network conditions used the amount of bandwidth used by the OpenSim server is about TCP fair. The Mongoose client is able to calculate and send a TCP fair throttle value to the server.

The OpenSim server's throttle system throttles each channel to the amount of bandwidth it is asked to by the client, this results in the total bandwidth used being less than the total throttle value. This is because some channels do not use the bandwidth that they are assigned and this unused bandwidth cannot be allocated to other channels and so is lost to the system. The modified OpenSim server created is able to share out unused bandwidth from the channels not using it to the channels that have packets that are able to use it. This modified server is able to limit the bandwidth to the value sent from the server. The Mongoose client is able to calculate a TCP fair bandwidth value and use this value to control the amount of bandwidth used.

Chapter 9

Summary and Conclusions

This chapter summarises the thesis and draws conclusions: the extent to which the goals have been achieved and the extent to which the thesis statement has been demonstrated is evaluated. The nature of computer networks and the problem of network congestion that arises in those networks is discussed. The design and function of TCP congestion control is described, followed by a description the long term properties of this system and the properties of Multi User Virtual Environments (MUVE) are described. This allows the topic of MUVE traffic management to be described.

The goals of this thesis are then stated followed by a description and summary of the contents of the chapters and how the chapters achieve or describe the achievement of the goals. The achievement of the goals is then evaluated. The contributions of the work are described. The thesis statement of the document is given and a description and discussion of the extent to which it has been demonstrated is presented. This chapter is arranged into the following sections:

Section 9.1: A description of the properties of the traffic management system.

Section 9.2: The extent to which the goals have been achieved is discussed.

Section 9.3: The contributions of the work.

Section 9.5: The extent to which the thesis statement has been demonstrated is discussed.

Section 9.6: The conclusions.

9.1 Traffic management properties

A Multi User Virtual Environment (MUVE) [131, 176, 115, 117, 3, 62, 92] in this work refers to a 3D virtual environment where multiple users can interact with each other and the environment through avatars. MUVEs are similar to but different from 3D multi player games. The environment provided is not fixed and can be changed by the user usually from inside the environment. The defining properties of MUVEs are:

- Realistic: the environment being is simulated is similar to the real world.
- Persistent: the environment persists over long periods of time and change to the environment remain.
- Mutability: the environment can be changed by the users.
- No objective: there are no predefined objectives provided by the environment.
- 3D: the environment is 3D.
- Avatar: the user interacts with the environment through an avatar.
- Distributed: the users can be physically distributed in the real world.

The physical distribution of MUVEs means that there is network communication between the clients and servers. The mutability of the environment means that the network traffic must contain the information about the environment.

Packet switched networks [67] such as the Internet have flexibility to allow for many different application types. The Internet is a datagram network [159] with no prior resource allocation, with inherently bursty traffic. This can allow cheap and efficient utilization of resources by multiplexing of bursty applications. The cost of this is network congestion, when the buffers in the router fill up and packets are lost. When packets are lost they will often need to be re-sent this can lead to more traffic which increases congestion and packet loss, this can lead in an uncontrolled network to congestion collapse where the majority of the resources of the network are being used for packet retransmission. As a result a congestion control system that prevents congestion collapse is necessary.

The Internet is a global packet switched network using a layered model. The four levels of the layered model [17] are Application, Transport, Internet and Link. The common feature is the Internet layer that uses the Internet Protocol [159] (IP). The most common transport layer protocol is Transmission Control Protocol [160] (TCP). There were congestion collapse events in the 1980's [133] on the Internet. As a result of this congestion control was created to control congestion on the Internet and prevent congestion collapse. TCP is a reliable in order transport protocol, it is not appropriate for all types of traffic so it is not used for all types of traffic.

TCP is used for the majority of the traffic on the Internet. It is appropriate for traffic that requires reliable data transport. TCP contains a congestion control system that has protected the Internet from congestion collapse. TCP's congestion control system converges to a fair allocation of bandwidth. Congestion control algorithms are said to be TCP friendly if they use less bandwidth than TCP in the worst case. The formula 2.3 gives the long term amount of bandwidth used by TCP with a given round trip time and loss rate. TCP Friendly Rate Control [72] uses a more complex formula that models the specific behaviour of TCP RENO from [146] by Padhye et al.. Not all of the congestion control systems that have this long term packet transmission rate also respond to congestion at the same rate and by the same amount. Bansal and Balakrishnan in [9] look at congestion control systems that have the same formula for their long term congestion control and compete fairly with TCP under the correct conditions. When the routers use Random Early Drop (RED) this system competes fairly with TCP, however when other policies are used for dropping packets at the routers these congestion window evolution formulae are not guaranteed to converge with those used by TCP. These congestion control systems will also converge when competing with TCP or any of this family of systems.

The TCP fair formula gives the bandwidth utilisation for long lasting connections, where the connection spends most of its time in the steady state. A TCP connection starts in the slow start mode and continues in this mode until it encounters loss or the slow start window is reached. The formula gives information about the congestion window however TCP connections are also constrained by the flow control window. The flow control window indicates the amount of data that one end of the connection can accept from the other, it is not affected by congestion or other network conditions, though some implementations will start it at a small value and increase it over time during the initial slow start phase of the connection. Connections that never reach their fair share of the bandwidth or their flow control window, as a result of being short lived and not involving much data, may experience no loss and so are limited in their in bandwidth utilisation only by the round trip time. The conditions on computer networks change with time, both on a long time scale and on a short time scale. It is important for a congestion control system to be able to adapt to changing network conditions. When the conditions change it takes time for the connections to reach their steady state.

Applications that do not use TCP and have substantial amount of bandwidth usage need to implement a congestion control system. These systems can be more complex than TCP's congestion control system, or as simple as a set sending rate. A sending rate that can be adjusted will provide some the ability to adapt to network conditions. A connection that is TCP fair uses a comparable amount of bandwidth as TCP under similar network conditions and react in ways that are compatible with TCP's congestion control.

The traffic of MUVEs is different [53] from that of other types of traffic. Its timing constraints are such that TCP is not normally considered appropriate for transporting its data, so the User Datagram Protocol [156] (UDP) is used. UDP provides a packet based semantics, it does not provide reliability or congestion control. The amount of bandwidth used by MUVEs is large and can put significant stress on the network. Second Life [115] is an example of a MUVE, which can use up to 1500 of kbps per user. It is important for MUVEs to be fair to TCP traffic. The traffic of virtual worlds are similar to multi user games[12] in that there are multiple users interacting in a virtual world and

[1]Blizzard Entertainment. World of Warcraft [Computer game]. `http://us.battle.net/wow/en/`, 23 November 2004. [Online; accessed 04-April-2011]

[2]ShenZhou Online `http://www.ewsoft.com.tw/` [Online; accessed 03-October-2010]

there is traffic for the avatar movement, however the environment is not changing in games so their traffic is much less. The traffic of MUVEs differ from that of video and audio streaming [80] because MUVE traffic is bursty. When an avatar moves around the environment it encounters objects and textures which are then downloaded. An important component of MUVE traffic is this downloading of images and objects, the faster they can be downloaded the faster they can be shown to the user, this means that MUVE traffic is naturally bursty.

Second Life and OpenSim [131] have traffic management schemes that adjust the limit on their bandwidth usage based on the level of congestion. They also distribute bandwidth between channels which separate the different types of packet that have different functions, however the mechanism used to determine the rate unnecessarily limits the rate at low loss levels and the limit is too high at medium to high loss levels. Furthermore the allocation to traffic types is not optimal as avatar traffic, which is critical, is mixed with other less critical types of object traffic.

We have shown that by making some relatively simple changes to the client it is possible to determine a TCP fair rate over the range of loss regimes that the application will encounter. Further it is possible to ensure that by making changes to the client it is possible to ensure that the system reacts in a TCP fair way to congestion. We have shown that it is possible with the OpenSim server provided that adjustments are made to how the server limits to bandwidth usage based on the provided value. The OpenSim server separates out the avatar traffic protecting its bandwidth from the other traffic types. In these ways we have shown that MUVEs can be engineered so that they can be made TCP fair and the avatar traffic, which is important, can be protected.

9.2 Summary of Work

This section will summarise the work. Firstly by looking at the goals of this work. Then by giving a summary of the contents and achievements of the chapters. Finally, summarise the achievement of the goals.

9.2.1 Goals

The goals are as stated in the introductory chapter. The first stage of this work was to develop an understanding of MUVE traffic. This understanding was then used to design a traffic management system that is able to make use of network resources to meet user needs and adapt to network conditions. The following goals were identified and met:

1. Through analysis of related work to contribute to the understanding of Multi User Virtual Environment's traffic, placing MUVEs into context.
2. Measurement and experimentation to contribute to the understanding of Multi User Virtual Environment's traffic.
3. To design adaptive congestion control for MUVE which:
   (a) Adapts to changes in network conditions at time-scales appropriate for application utility and network stability.
   (b) Competes fairly with TCP and other Internet traffic and is therefor able to receive an appropriate share of network resources.
4. To implement adaptive congestion control system in a modified Second Life client, allowing it to be used with OpenSim and Second Life servers.
5. To evaluate the implementation and the design, by comparing with standard SL behaviour and standard TCP fair behaviour.

9.2.2 Summary of Chapters

The topic of this dissertation is the traffic of Multi User Virtual Environments (MUVEs) and MUVEs are systems which simulate environments that are similar to the real world, which the user interacts with through an avatar. The goal of creating an adaptive traffic management system that meets the need of MUVEs has been described. There are differences between MUVE traffic and other network traffic that require them to have their own traffic management system and not to use the TCP one. This chapter lays out the rest of the document.

Chapter 2 describes the Internet. The design of the Internet as a packet switched network using a layered system where each layer use the services provided by the layer below and provides services to the layer above. The function of routers is to forward packets to their destination and the problem of congestion where packets are lost when too many are sent onto the network was described. The IP protocol is used to transport the packets on the Internet using a number of different lower layer protocols. The majority of the packets on the Internet use TCP as their transport layer protocol. TCP provides a reliable, in order, byte stream abstraction. Reliable protocols can cause congestion collapse where packets are lost and then re-sent causing more congestion, to the point where all of the packets are resends. TCP is congestion controlled which prevents the congestion collapse of the Internet and allows different connections to fairly share the available bandwidth.

The mechanism of congestion control is described where congestion is detected and the congestion control system adjusts in response. TCP uses a window to control the number of packets that are outstanding on the network, where packet loss is used as an indication of congestion. The window is increased additively in the absence of congestion reduced by half in the presence of congestion. The reliable nature of TCP means that when packets are lost they must be re-sent before the application can get access to the data after the lost part. This makes TCP not appropriate for MUVEs as they require to get information about avatar movement promptly.

The key points of this chapter are the understanding of the Internet, TCP and congestion control. This contributes to goals 3a and 3b of designing the traffic management system that is able to coexist with other Internet traffic.

Chapter 3 describes the methodology used for undertaking this work and the tools used. The process used when undertaking the work is described. TCP and its congestion control was investigated followed by DCCP and then MUVEs generally were investigated and Second Life in particular and then an adaptive traffic management system was design, implemented and tested.

The methods of packet capture that were used are described. A user space implementation of DCCP that was created is described. DCCP is a transport level protocol that provides congestion control, without reliability and does not require in order delivery of data to user space. An analysis program that was created to analyse DCCP and Second Life packets, is described, this program can generate statistics from the traces. The netem queue discipline that was used to change network conditions on the test network was described.

Chapter 4 describes MUVEs. MUVEs simulate an environment that is similar to the real world, that users can become immersed in. The simulated environment is similar to the real world in that the physics simulated is consistent with the real world. In this document MUVE does not refer to games. The history of MUVEs is described and MUVEs are compared to multi player games.

The MUVE Second Life, which this document focuses on, is described in detail. The user controls the avatar which is simulated by the server for the island the avatar is on. Second Life uses an application level protocol transported using the UDP transport level protocol. The packets between machines are arranged into connection like constructs called circuits, where the packets have sequence numbers that are used to detect loss. There are a number of different types of packets that are used for different purposes. Second Life has a system to control the rate that packets are sent to the client from the server. The rate value is calculated at the client and sent to the server, this rate calculation system is not TCP fair. It has an additive increase and the additive decrease adaptation mechanism, this means that it won't compete fairly with TCP or be able to converge with other implementations of its self.

The requirements that Second Life would have on a traffic management system are, that it must allow the timely delivery of packets, adapt to network conditions and allow the system to allocate sufficient bandwidth to different channels. The adaptation system must use congestion control that is TCP fair.

Key points of this chapter are a review of the literature regarding MUVEs and Second Life. This contributes to goal 1, of understanding MUVE traffic from the literature. This chapter describes MUVEs as have the five properties of: Realistic, Persistence, Mutability of environment, User control over objectives, 3D, Avatar and Distributed. The user control of objectives differentiate games from MUVEs. The environment of MUVEs is not predefined which is also different from games.

Chapter 5 describes the investigation of traffic of Second Life. The areas investigated were the relationship between the avatars actions and the traffic, the organization of the packets into circuits and channels in those circuits and the global levels of traffic. The traffic of a group of thirty users in a lab using Second Life was captured and analysed. From this it was determined that it was possible to have a lab of users using Second Life simultaneously and that it was possible for this situation to no use more bandwidth than its fair share. There is a relationship between the

activity undertaken and the environment it is undertaken in and the traffic generated by Second Life. The Second Life throttles control the amount of bandwidth used by Second Life. The majority of the packets are not full segments. This leads to the conclusion that a system which produces a rate which is TCP fair could be integrated into the Second Life client to make its traffic TCP fair.

Second Life traffic can be decomposed into channels which contain packets with different functions and require different amounts of bandwidth and have different packet size distributions. Avatar control packets are a small proportion of the total traffic (less than 10 Kbps) but have a disproportionate effect on system usability. This part of the chapter achieves goal 2 of learning about Second Life traffic from measurement.

To further investigate Second Life's traffic a series of controlled experiments were performed where a single user performed a set action for ten minutes on two different islands in Second Life with different throttle settings and network conditions. The client was modified to allow the difference between the client's estimate of the avatars position and the positions it receives from the server. The results of this work were consistent with previous studies. This is discussed in section 5.3.2. The throttle and state of the cache have an effect on the traffic and accuracy of the avatar's position. This part of the chapter achieves goal 2, of learning about Second Life traffic from experimentation. This chapter further contributes to goal 1.

Chapter 6 describes the design of an adaptive traffic management system for Second Life, based on a modified client connecting to the existing Second Life servers. The name of this modified client is Mongoose. The different types of network or network conditions that a user might experience where it would be necessary for Second Life to use less traffic are described and that using more bandwidth would allow for a better user experience where network conditions allow.

The Second Life throttle adaptation system was compared to TCP's congestion control system. TCP congestion control was described and TCP fair behaviour defined. The formula that gives long term bandwidth utilization of a network limited TCP connection is given. The operation of Second Life's throttle adaptation system was described and the region where Second Life is TCP fair presented.

The different types of congestion control that could be used were discussed. There are window based and rate based congestion control systems, window based system control the number of packets, or bytes, from a connection on the network and rate based system control the rate at which packets, or bytes, are sent out on to the network. The design of only modifying only the client limited Mongoose to using a rate based system. The value used by the congestion control system can be calculated using two methods, a window tracking system where a window is maintained and its value changed in reaction to congestion and non congestion, and equation based where a value is calculated based on statistics for the recent history of the connection. Window tracking systems adapt to changes in the network conditions more quickly than equation based systems.

The structure of the Second Life client is described as well as the process of its operation. The modifications that were performed in the creation of the Mongoose client are described. Mongoose uses the same RTT calculation as the Second Life client. The parts that process the sequence numbers of incoming packets and the part that performs the update to the throttle value were modified. The key point of this chapter is a design of an adaptive congestion control system for MUVEs achieving goals 3a and 3b of adapting to network conditions and competing fairly with TCP. This chapter also describes the implementation of this system achieving goal 4.

Chapter 7 presents an evaluation of the mongoose client. The throttle values calculated by the client were compared to the TCP fair values and the Second Life client's throttle values. This showed that window tracking system implemented in Second Life was able to produce a TCP fair rate. TFRC values were calculated from loss values extracted from the traces and these were compared to the TCP fair rate and Mongoose throttle values. Simulations of window tracking and equation based systems were performed that showed the equation based systems adapt more slowly to changes in network conditions than window tracking systems. Experiments where the loss rate was changed during a session were performed with the Mongoose client: these were consistent with the simulations. The affect of the throttle value on bandwidth utilization was evaluated and found to be working as well as the Second Life client. The bandwidth utilization was compared with TCP fair and found to be TCP fair. This chapter describes the evaluation of the designed system achieving goal 5.

Chapter 8 presents an evaluation of the traffic of the OpenSim server. The types and sizes of packets were investigated as well as the amount of TCP and UDP packets and bytes. The activities of Standing, Walking, Flying and Teleporting were investigated. The traffic produced by the OpenSim server was compared to that of the Second

Life server. The behaviour of the Second Life client with the OpenSim server under different network conditions was investigated. The throttle value calculated by the standard Second Life client is, like the Second Life server, not TCP fair. The Mongoose client was evaluated with the OpenSim server: it was found that it calculated a TCP fair rate.

The bandwidth used by the server per user and per channel was investigated. In addition the source of the throttle system was investigated. This investigation lead to the conclusion that its throttle system throttles each channel to the amount of bandwidth it is asked to by the client, which results in the total bandwidth used being less than the total throttle value. A modified version of the OpenSim server was created which allowed channels to use the bandwidth not used by other channels. The Mongoose client was evaluated with the modified OpenSim server. The throttle value sent from the client and the bandwidth usage by the server are TCP fair.

9.2.3 Summary

The goal of understanding MUVE traffic by analysing the literature has been accomplished. The literature is described in chapters 4 and 5. The context into which the MUVE traffic fits is described in chapter 2. The properties of the Internet have been described giving context for the properties of MUVE traffic. The properties of MUVE traffic have been described including their design and communication structure. This gives an understanding of the traffic of MUVEs and their control systems.

Measurement has been used to aid in the understanding of Second Life and OpenSim traffic. The measurement of Second Life is described in chapter 5. A live session of Second Life traffic was captured. This observation gave information about the nature of Second Life traffic, informing the experimental investigation. Experimentation has been used to further our understanding of Second Life traffic. This experimentation is described in chapter 5. The number of islands used in this experimentation was two. Each island is different so a larger number of islands would have given more information. OpenSim is also a MUVE, its traffic is similar to but different from that of Second Life. The measurement of OpenSim is described in chapter 8.

The goal of designing an adaptive congestion control system that competes fairly with TCP has been completed in that it reacts to network conditions in the same way as TCP. This system has been implemented and evaluated. The system could be further tested with a larger number of network conditions. The design of the window tracking system is based on the system used by TCP.

The goal of designing an adaptive congestion control system that adapts at the appropriate time scale has been accomplished. A modified client with the existing server limited the frequency with which the rate could be changed makes the system change its rate less often than it would otherwise need to.

The goal of implementing the designed adaptive traffic management system has been completed. This is described in chapter 6. This design has been implemented and evaluated achieving the goal of evaluating the implemented traffic management system, this is described in chapter 7. The Mongoose client has also been evaluated with the OpenSim server. The problem of the OpenSim server limiting the traffic more than the throttle value sent from the client has been fixed.

The properties of MUVEs have been investigated and the network requirements discovered. A traffic management system for Second Life and OpenSim has been designed, implemented and evaluated. The goals of this work have been achieved.

9.3 Contributions of the Work

At the top level the core contribution of this work is the design, implementation and evaluation of client side measurement based differential congestion control for MUVEs. The design was based upon a substantial analysis and measurement study of Second Life. In the process a number of subsidiary original contributions have been made:

- Analysis of traffic management: An analysis of network traffic management within the popular MUVE Second Life has been undertaken. This includes identification and definition of the function of the following components: throttles for controlling bandwidth, circuits for organising packet flows, packets for containing and identifying chunks of data, channels for differentiating between different types of data within circuits and congestion feedback. The analysis is based upon, existing system documentation, examination of SL Client and OpenSim server source code along with analysis of network traffic traces and is the first analysis to establish the relationships between the above components.

- Analysis of requirements: The observation that MUVE traffic can be decomposed into Avatar and Environment traffic, which have different requirements is made. This contrasts with the current Second Life decomposition which is organised by application functionality. To achieve this significant research into how over 400 packet types relate to Second Life channels was required.
- Development of measurement infrastructure:
  - A user-space implementation of the Datagram Congestion Control Protocol (DCCP), which is of value both in the investigation of congestion control algorithms for Second Life and for wider use.
  - A tool, called SLparse, for the analysis of Second Life traffic which is aware of Second Life packet types and how these relate to channels.
  - The development of an instrumented Second Life client.
  - The development of a Second Life Client which enables control over bandwidth requests from the client to the server both for the global throttle and for the channel throttles.
  - The development of a test network which enables control over models of packet loss, delay, and available bandwidth.
- Measurement: Past studies have created a number of "typical" scenarios. For example looking at the traffic generated by an Avatar, flying, walking and standing still in busy and quite areas. This work validates, builds upon and extends that work:
  - A study of traffic generated by users as they use Second Life in a real world setting is undertaken.
  - The interaction between traffic application behaviour, traffic management and network conditions is examined. Experiments to evaluate, Second Life congestion control and the appropriateness of rate and window based congestion control were conducted.
    - It is shown that under "normal" conditions Second Life is less aggressive than TCP and functions well.
    - The effectiveness of Second Life's mechanisms depend significantly on user settings.
    - Second Life does not adapt well to changes in network conditions.
    - When network resources are plentiful Second Life may unnecessarily constrain bandwidth usage.
    - There are scenarios where if congestion levels are substantial Second Life is overly aggressive in its use of bandwidth.
  - A comparison of Second Life traffic management system with TCP's congestion control system.
- Design: The design of the functional components of measurement based differential congestion control for MUVEs, which is based upon our analysis, measurement and experimentation of Second Life traffic.
  - An evaluation of the appropriateness of different types of congestion control algorithm for controlling Second Life traffic. The different methods of calculating the rate used by the traffic management, window tracking and equation based system have been evaluated. This led to the conclusion that window tracking system converge to the correct rate more quickly and are appropriate to MUVE traffic.
  - An evaluation of what can be done with a modified client and the existing server and investigation in to what could be accomplished by modifying both the client and server.
- Implementation: Previous work has focused on characterising Second Life traffic and making general architectural recommendations. We go further and design, implement and evaluate traffic control mechanisms for MUVEs.
- Evaluation: The evaluation of the Mongoose client. The ability of the client is evaluated and the performance of different algorithms. The ability of the client to control traffic and produce a TCP fair rate has been demonstrated.
- An evaluation of the importance of interval between updates to the rate calculated by the client are made and a determination of what the appropriate rate is.

- Measurement: A study of the traffic generated by the OpenSim server. Looking at the traffic generated by an Avatar, flying, walking, standing and teleporting on different islands. A comparison with the traffic of Second Life. This has shown that traffic application generation is similar to Second Life traffic.
- Measurement: An examination of the throttle system used by OpenSim to achieve the limits on the traffic asked for by the Second life client. This has shown that OpenSim's traffic management system is stricter than that of Second life's.
- Evaluation: An evaluation of the Mongoose client with the OpenSim server and the interaction of its throttle system with Mongoose's throttle calculation system. Which demonstrates that the Mongoose client is able to calculate a TCP fair throttle value and send it to the server.
- Implementation: The design and implementation of modifications to the OpenSim throttle system to allow it to share used throttle between its channels.
- Evaluation: An evaluation of the modifications made to OpenSim. This evaluation of the combinations of the client and servers has shown that:
  - The Second Life client and the unmodified OpenSim server – The client calculates a throttle value that is too large and the server limits the bandwidth to less than the throttle value.
  - The Mongoose client and the unmodified OpenSim server – The client calculates and sends a TCP fair throttle value and the server limits the bandwidth to less than the throttle value.
  - The Second Life client and the modified OpenSim server – The client calculates a throttle value that is too large and the server limits its bandwidth to this value.
  - The Mongoose client and the modified OpenSim server – The client calculates a TCP fair value and the server uses this value to limit the bandwidth to a TCP fair level.

This work contributes significantly to our understanding of Second Life traffic. It builds upon and is consistent with previous work on computer games and MUVEs. It presents a novel design, implementation and evaluation of traffic control for MUVEs which is based upon significant analysis, measurement and experimental studies.

9.4 Limitations of the Work and Future work

The investigation of Second Life and OpenSim involved resetting the client cache before each run, this allowed the experiments to be repeated. This is not consistent with the behaviour of all users and the cached objects would effect the traffic. There was an investigation of the effect of the caching on the traffic and of traffic conditions on the traffic but not the effect of cache on the behaviour under different network conditions. An investigation of Second Life traffic with different levels cache should be possible by replacing the cache files with saved versions for different levels of cache.

There were a small number of Islands used that had different properties, these were existing islands. The avatars moving around the Islands in Second Life were users. This methodology aloud the experiments to be repeated but did not allow a larger range of possible islands to be investigated. It would be possible to create a large number of different Islands with different properties.

The behaviour of Second Life and OpenSim where the throttle value is large is that the server uses less bandwidth than the throttle value. We hypothesise that this is the result of application limitation in that the server does not have more traffic to send. The traffic is consistent in that in becomes more bursty when the throttle value is higher and the mean bandwidth will not increase over a certain value. This could be tested by creating islands with larger numbers of objects and textures and having a large number of avatars on them, this would result in the server having more traffic to send.

The SLparse program was implemented using the PCAP library to read traces. This required that the parser to be implemented from the start. It would have been possible to attempt to use or adapt an existing program for parsing Second Life protocol packets and to generate the graphs and statistics.

The USDL library and the Mongoose clients have congestion control systems using there own implementations of these algorithms. It would have been possible to using existing library implementations to implement these congestion control systems.

The error in the location of the avatars was recorded with a modified Second Life client with the Second Life server. The effect this has on users' utility fro the application was not studied. The effect of the Mongoose client on the positional error was not studied. The positional error was not studied with the OpenSim server. The importance to the users of positional error could be studied with a group of users. The modified client for detecting the positional error could be used with the OpenSim server to study the positional error of OpenSim. A modified version of the Mongoose client could be created to detect the position errors and the effect of Mongoose's throttle adaptation system.

9.5 Thesis statement

The thesis statement in this dissertation is:

The thesis of this work is that measurement based adaptive congestion control will allow MUVE traffic to coexist fairly with established Internet traffic whilst improving the service the application receives from the network. Such a system can be achieved through:

1. *A system limited to the client side and therefore usable with the current Second Life and OpenSim servers.*
2. *A window tracking rate algorithm to allow adaptation to network conditions.*

When compared to current MUVE traffic management the system will be TCP fair over a wider range of network conditions. When compared with equation based algorithms systems, such as TCP friendly rate control (TFRC), it will allow quicker adaptation to changes in network conditions.

The system designed, implemented and tested is implemented in a modified client that is able to connect and work with the unmodified Second Life and OpenSim servers. The Mongoose client uses a window tracking system that is able to adapt to different network conditions. The rate produced by this system is TCP fair under wider range of network conditions than the system implemented by Second Life. This window tracking system has been compared to TFRC and been found to adapt more quickly.

The evaluation described in chapter 7 shows that the system is able to produce a rate similar to TCP. The design of this system is described in chapter 6. The designed system uses a modified client and is able to connect to the existing Second Life and OpenSim servers. The designed and implemented client uses a window tracking algorithm which has been shown to be able to adapt to network conditions. The evaluated with OpenSim described in chapter 8 discovered that the client is able to send a TCP fair throttle value but the server will limit its bandwidth usage to less than this value. A modification allowed the server to use its TCP fair share. The unmodified server is able to work with the Mongoose client and its bandwidth usage is TCP fair though it uses less bandwidth than its TCP fair share.

The designed system is TCP fair under a wider range of network conditions than the existing Second Life system. The evaluation in chapter 7 has shown that window tracking systems in general and this system in particular is able to more quickly adapt to network conditions than existing systems. The thesis statement has been validated with the qualification that modifications to OpenSim server were required.

9.6 Conclusion

The areas relating to MUVEs and traffic management have been described. I have described the Internet and network protocols, the analysis of the operation of network protocols and the design of network protocols. The significant differences between MUVE and multi user computer games have been described.

The project has been substantial involving investigating different types of congestion control and the operation of networks. It also involved understanding the properties and operation of MUVEs. A traffic management system has been designed and implemented based on this understanding of MUVEs and network traffic.

The methodology is sound, employing a variety of appropriate techniques. Traffic has been captured outside of controlled conditions and experiments have been designed and performed. Network systems have been statically analysed and simulations of network conditions and control systems have been used. These simulations have been compared with experiments. These experiments have shown that the designed system is able adapt to network conditions and is TCP fair.

The work is new and innovative in that to my knowledge no one else has addressed TCP fairness in a MUVE system. It is significant and important because MUVEs are a new and widely used technology.

The goal of under standing the properties of MUVEs and creating a traffic management system for Second Life and OpenSim has been achieved. The creation and evaluation of this system has demonstrated the thesis of this work.

Bibliography

[1] Duran duran to give virtual gigs. `http://news.bbc.co.uk/1/hi/technology/5253782.stm`, 8 August 2006. [Online; accessed 01-October-2010].

[2] M. Accetta, R. Baron, W. Bolosky, D. Golub, R. Rashid, A. Tevanian, and M. Young. Mach: A new kernel foundation for UNIX development. *contract*, 39(85-C):1034.

[3] Active Worlds, Inc. Active Worlds. `http://www.activeworlds.com/`, 1997. [Online; accessed 29-September-2010].

[4] M. Allman, V. Paxson, and W. Stevens. TCP Congestion Control. RFC 2581 (Proposed Standard), April 1999. Updated by RFC 3390.

[5] Bjorn Andersson, Erik Ekman, and Anne Bezemer. Iodine. `http://code.kryo.se/iodine/`, February 2010. [Online; accessed 29-March-2011].

[6] Rafael Antonello, Stnio Fernandes, Josilene Moreira, Paulo Cunha, Carlos Kamienski, and Djamel Sadok. Traffic Analysis and Synthetic Models of Second Life. *Multimedia Systems*, 15:33–47, 2009. 10.1007/s00530-008-0125-1.

[7] F. Baker. Requirements for IP Version 4 Routers. RFC 1812 (Proposed Standard), June 1995. Updated by RFC 2644.

[8] H. Balakrishnan and S. Seshan. The Congestion Manager. RFC 3124 (Proposed Standard), June 2001.

[9] Deepak Bansal and Hari Balakrishnan. Binomial congestion control algorithms. In *INFOCOM 2001. Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 2, pages 631–640, 2001.

[10] S. Barab, M. Thomas, T. Dodge, R. Carteaux, and H. Tuzun. Making learning fun: Quest Atlantis, a game without guns. *Educational Technology Research and Development*, 53(1):86–107, 2005.

[11] Tom Beigbeder, Rory Coughlan, Corey Lusher, John Plunkett, Emmanuel Agu, and Mark Claypool. The effects of loss and latency on user performance in Unreal Tournament 2003 ®. In *NetGames '04: Proceedings of 3rd ACM SIGCOMM workshop on Network and system support for games*, pages 144–151, New York, NY, USA, 2004. ACM.

[12] P. Benko and A. Veres. A passive method for estimating end-to-end TCP packet loss. In *Global Telecommunications Conference, 2002. GLOBECOM '02. IEEE*, volume 3, pages 2609 – 2613 vol.3, nov. 2002.

[13] Laurent Bernaille, Renata Teixeira, Ismael Akodkenou, Augustin Soule, and Kave Salamatian. Traffic classification on the fly. *SIGCOMM Comput. Commun. Rev.*, 36(2):23–26, 2006.

[14] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss. An Architecture for Differentiated Service. RFC 2475 (Informational), December 1998. Updated by RFC 3260.

[15] Manfred Bogen, Michael Lenz, Andreas Riechpietsch, and Peter Simons. Securing ordinary tcp services through tunnels. `http://cryp.to/publications/tunneling-ordinary-tcp-services/`, April 17 1998. [Online; accessed 04-April-2011].

[16] R. Bonica, D. Gan, D. Tappan, and C. Pignataro. Extended ICMP to Support Multi-Part Messages. RFC 4884 (Proposed Standard), April 2007.

[17] R. Braden. Requirements for Internet Hosts - Communication Layers. RFC 1122 (Standard), October 1989. Updated by RFCs 1349, 4379.

[18] R. Braden, D. Clark, and S. Shenker. Integrated Services in the Internet Architecture: an Overview. RFC 1633 (Informational), June 1994.

[19] R. Braden, L. Zhang, S. Berson, S. Herzog, and S. Jamin. Resource ReSerVation Protocol (RSVP) – Version 1 Functional Specification. RFC 2205 (Proposed Standard), September 1997. Updated by RFCs 2750, 3936, 4495, 5946.

[20] Lawrence S. Brakmo, Sean W. O'Malley, and Larry L. Peterson. TCP Vegas: new techniques for congestion detection and avoidance. *SIGCOMM Comput. Commun. Rev.*, 24(4):24–35, 1994.

[21] T. Braun, C. Diot, A. Hoglander, and V. Roca. An experimental user level implementation of TCP. *Technical Report, INRIA Sophia Antipolis, France*, (2650), 1995.

[22] Ron Britvich. WebWorld. Peregrine Systems Inc., 1994.

[23] Jason But, Urs Keller, and Grenville Armitage. Passive TCP Stream Estimation of RTT and Jitter Parameters. In *LCN '05: Proceedings of the The IEEE Conference on Local Computer Networks 30th Anniversary*, pages 433–441, Washington, DC, USA, 2005. IEEE Computer Society.

[24] Kuan-Ta Chen, Polly Huang, Chun-Ying Huang, and Chin-Laung Lei. Game traffic analysis: an MMORPG perspective. In *NOSSDAV '05: Proceedings of the international workshop on Network and operating systems support for digital audio and video*, pages 19–24, New York, NY, USA, 2005. ACM.

[25] Kuan-Ta Chen, Polly Huang, and Chin-Laung Lei. How sensitive are online gamers to network quality? *Commun. ACM*, 49(11):34–38, 2006.

[26] PF Chimento. Standard Token Bucket Terminology. *Technical documentation* `http://qbone.internet2.edu/bb/Traffic.pdf`, 2000.

[27] D.-M. Chiu and R. Jain. Analysis of the increase and decrease algorithms for congestion avoidance in computer networks. *Comput. Netw. ISDN Syst.*, 17(1):1–14, 1989.

[28] K. Cho. WIDE Update. `http://www.caida.org/workshops/wide-casfi/1004/slides/wide-casfi1004_kcho.pdf`, 2010. [Online; accessed 04-April-2011].

[29] KC Claffy and S. McCreary. Internet measurement and data analysis: passive and active measurement. *University of California, CAIDA, USA* `http://www.caida.org/publications/papers/1999/Nae4hansen/`, 1999.

[30] B. Claise. Cisco Systems NetFlow Services Export Version 9. RFC 3954 (Informational), October 2004.

[31] D. Clark. The design philosophy of the DARPA Internet protocols. *ACM SIGCOMM Computer Communication Review*, 18(4):106–114, 1988.

[32] D. D. Clark and D. L. Tennenhouse. Architectural considerations for a new generation of protocols. In *SIGCOMM '90: Proceedings of the ACM symposium on Communications architectures & protocols*, pages 200–208, New York, NY, USA, 1990. ACM Press.

[33] David D. Clark, John Wroclawski, Karen R. Sollins, and Robert Braden. Tussle in cyberspace: defining tomorrow's internet. *IEEE/ACM Trans. Netw.*, 13(3):462–475, 2005.

[34] M. Claypool. The effect of latency on user performance in real-time strategy games. *Computer Networks*, 49(1):52–70, 2005.

[35] Mark Claypool and Kajal Claypool. Latency and player actions in online games. *Commun. ACM*, 49(11):40–45, 2006.

[36] Fivos Constantinou and Panayiotis Mavrommatis. Identifying known and unknown peer-to-peer traffic. In *NCA '06: Proceedings of the Fifth IEEE International Symposium on Network Computing and Applications*, pages 93–102, Washington, DC, USA, 2006. IEEE Computer Society.

[37] Digital Equipment Corporation. The ethernet: a local area network: data link layer and physical layer specifications. *SIGCOMM Comput. Commun. Rev.*, 11(3):20–66, 1981.

[38] JP Coudreuse, G. Pays, and M. Trouvat. Asynchronous transfer mode. *Commutation and Transmission*, 3:5–16, 1990.

[39] B.P. Crow, I. Widjaja, L.G. Kim, and P.T. Sakai. IEEE 802.11 Wireless Local Area Networks. *Communications Magazine, IEEE*, 35(9):116 –126, September 1997.

[40] William Crowther, Don Woods, and K. Black. Colossal cave adventure. Published by Cambridge, MA: Bolt, Beranek and Newman, 1976.

[41] M. Deering and H. Sowizral. Java3D Specification, Version 1.0. *Sun Microsystems*, 2550.

[42] S. Deering and R. Hinden. Internet Protocol, Version 6 (IPv6) Specification. RFC 2460 (Draft Standard), December 1998. Updated by RFCs 5095, 5722, 5871.

[43] EW Dijkstra. Making a Translator for ALGOL 60. *APIC Bull*, 7:3–11, 1961.

[44] Peter A. Dinda. The Minet TCP/IP Stack. Technical report, Department of Computer Science, Northwestern University, 2002.

[45] M. Duke, R. Braden, W. Eddy, and E. Blanton. A Roadmap for Transmission Control Protocol (TCP) Specification Documents. RFC 4614 (Informational), September 2006.

[46] ECMA. *ECMA-334: C# Language Specification*. ECMA (European Association for Standardizing Information and Communication Systems), pub-ECMA:adr, fourth edition, June 2006.

[47] David Ely, Stefan Savage, and David Wetherall. Alpine: a user-level infrastructure for network protocol development. In *USITS'01: Proceedings of the 3rd conference on USENIX Symposium on Internet Technologies and Systems*, pages 15–15, Berkeley, CA, USA, 2001. USENIX Association.

[48] Gert Svensson Emil Brink, Eskil Steenberg. The verse networked 3d graphics platform. *SIGRAD 2006. The Annual SIGRAD Conference, Special Theme: Computer Games. November 22–23, 2006, Skövde, Sweden*, 19(9):5, November 2006.

[49] T.L. Fanderclai. MUDs in education: New environments, new pedagogies. *Computer-Mediated Communication Magazine*, 2(1):8, 1995.

[50] W.-C. Feng, D.D. Kandlur, D. Saha, and K.G. Shin. A self-configuring red gateway. In *INFOCOM '99. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 3, pages 1320 –1328 vol.3, mar 1999.

[51] Wu-chang Feng, Francis Chang, Wu-chi Feng, and Jonathan Walpole. Provisioning on-line games: a traffic analysis of a busy counter-strike server. In *IMW '02: Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurment*, pages 151–156, New York, NY, USA, 2002. ACM.

[52] Wu-chang Feng, Francis Chang, Wu-chi Feng, and Jonathan Walpole. A traffic characterization of popular on-line games. *IEEE/ACM Trans. Netw.*, 13(3):488–500, 2005.

[53] Stenio Fernandes, Fadael Antonello, Josilene Moreira, Djamel Sadok, and Carlos Kamienski. Traffic Analysis Beyond This World: The Case of Second Life. In *7th International workshop on Network and Operating Systems Support for Digital Audio & Video (NOSSDAV) 2007*, 2007.

[54] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1. RFC 2616 (Draft Standard), June 1999. Updated by RFC 2817.

[55] S. Floyd, M. Handley, and E. Kohler. Problem Statement for the Datagram Congestion Control Protocol (DCCP). RFC 4336 (Informational), March 2006.

[56] S. Floyd and V. Jacobson. Random Early Detection Gateways for Congestion Avoidance. *IEEE/ACM Transactions on Networking*, 1(4):397–413, 1993.

[57] S. Floyd and E. Kohler. Profile for Datagram Congestion Control Protocol (DCCP) Congestion Control ID 2: TCP-like Congestion Control. RFC 4341 (Proposed Standard), March 2006.

[58] S. Floyd, E. Kohler, and J. Padhye. Profile for Datagram Congestion Control Protocol (DCCP) Congestion Control ID 3: TCP-Friendly Rate Control (TFRC). RFC 4342 (Proposed Standard), March 2006.

[59] Sally Floyd, Mark H, and Jitendra Padhye. A comparison of equation-based and aimd congestion control. Technical report, ACIRI, May 2000.

[60] Eelke Folmer, Bei Yuan, Dave Carr, and Manjari Sapre. TextSL: a command-based virtual world interface for the visually impaired. In *Proceedings of the 11th international ACM SIGACCESS conference on Computers and accessibility*, Assets '09, pages 59–66, New York, NY, USA, 2009. ACM.

[61] M. Fomenkov, K. Keys, D. Moore, and K. Claffy. Longitudinal study of Internet traffic in 1998-2003. In *Proceedings of the winter international synposium on Information and communication technologies*, pages 1–6. Trinity College Dublin, 2004.

[62] Open Wonderland Foundation and Sun Microsystems. Open Wonderland. `http://openwonderland.org/`. [Online; accessed 29-September-2010].

[63] Kristoffer Getchell. *Enabling exploratory learning through virtual fieldwork*. PhD thesis, University of Standrews, October 2009.

[64] Kristoffer Getchell, Alan Miller, Colin Allison, and Rebecca Sweetman. *Serious Games on the Move*, chapter Exploring the Second Life of a Byzantine Basilica, pages 165–180. SpringerWien, 2009.

[65] Kristoffer Getchell, Iain Oliver, Alan Miller, and Colin Allison. Metaverses as a platform for game based learning. In *Proceedings of the 2010 24th IEEE International Conference on Advanced Information Networking and Applications*, AINA '10, pages 1195–1202, Washington, DC, USA, 2010. IEEE Computer Society.

[66] L. Gharai. RTP with TCP Friendly Rate Control. *draft-ietf-avt-tfrc-profile-10 (work in progress)* *http://tools.ietf.org/html/draft-ietf-avt-tfrc-profile-10*, 22 July 2007. Work in progress.

[67] C.H. Giancarlo. Packet switching network, August 30 1988. US Patent 4,768,190.

[68] James Gosling, Bill Joy, Guy Steele, and Gilad Bracha. *The Java Language Specification, Third Edition*. Addison Wesley, 3rd edition, 2005.

[69] Audio-Video Transport Working Group and H. Schulzrinne. RTP Profile for Audio and Video Conferences with Minimal Control. RFC 1890 (Proposed Standard), January 1996. Obsoleted by RFC 3551.

[70] James Hall, Andrew Moore, Ian Pratt, and Ian Leslie. Multi-protocol visualization: a tool demonstration. In *MoMeTools '03: Proceedings of the ACM SIGCOMM workshop on Models, methods and tools for reproducible network research*, pages 13–22, New York, NY, USA, 2003. ACM.

[71] James Hall, Ian Pratt, and Ian Leslie. Observing web browser behaviour using the nprobe passive monitoring architecture (extended abstract).

[72] M. Handley, S. Floyd, J. Padhye, and J. Widmer. TCP Friendly Rate Control (TFRC): Protocol Specification. RFC 3448 (Proposed Standard), January 2003.

[73] S. Hemminger. Network emulation with NetEm. In *Linux Conf Au* *http://devresources.linuxfoundation.org/shemminger/netem/LCA2005_paper.pdf*, 2005.

[74] Tristan Henderson. Latency and user behaviour on a multiplayer game server. In *NGC '01: Proceedings of the Third International COST264 Workshop on Networked Group Communication*, pages 1–13, London, UK, 2001. Springer-Verlag.

[75] Tristan Henderson and Saleem Bhatti. Networked games: a QoS-sensitive application for QoS-insensitive users? In *RIPQoS '03: Proceedings of the ACM SIGCOMM workshop on Revisiting IP QoS*, pages 141–147, New York, NY, USA, 2003. ACM.

[76] D. Hoffman, G. Fernando, V. Goyal, and M. Civanlar. RTP Payload Format for MPEG1/MPEG2 Video. RFC 2250 (Proposed Standard), January 1998.

[77] C. Hornig. A Standard for the Transmission of IP Datagrams over Ethernet Networks. RFC 894 (Standard), April 1984.

[78] Gianluca Iannaccone, Christophe Diot, Derek McAuley, and Luigi Rizzo. The CoMo White Paper, September 2004.

[79] Dan Ingalls, Ted Kaehler, John Maloney, Scott Wallace, and Alan Kay. Back to the future: the story of squeak, a practical smalltalk written in itself. In *OOPSLA '97: Proceedings of the 12th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, pages 318–326, New York, NY, USA, 1997. ACM.

[80] International Telecommunication Union. Packet-based Multimedia Communications Systems. *Telecommunication Standardization Sector of ITU*, July 2003.

[81] V. Jacobson. Congestion Avoidance and Control. *ACM SIGCOMM Comput. Commun. Rev.*, 18(4):314–329, 1988.

[82] V. Jacobson, C. Leres, and S. McCanne. The tcpdump manual page. *Lawrence Berkeley Laboratory, Berkeley, CA*, 1989.

[83] V. Jacobson, C. Leres, and S. McCanne. pcap-Packet Capture library. *UNIX man page*, 1993.

[84] Van Jacobson. Modified TCP congestion avoidance algorithm. `ftp://ftp.isi.edu/end2end/end2end-interest-1990.mail` end2end-interest mailing list, 30 April 1990. [Online; accessed 03-November-2010].

[85] R. Jain. A Timeout-Based Congestion Control Scheme for Window Flow-Controlled Networks. *EEE Journal on Selected Areas in Communications*, 4(7):1162–1167, Oct 1986.

[86] R. Jain, D. M. Chiu, and W. Hawe. A quantitative measure of fairness and discrimination for resource allocation in shared systems. Technical Report DEC TR-301, Digital Equipment Corporation, Littleton, MA, 1984.

[87] Raj Jain. Congestion control in computer networks: issues and trends. *IEEE Network*, pages 24–30, 1990.

[88] Hao Jiang and Constantinos Dovrolis. Passive estimation of tcp round-trip times. *SIGCOMM Comput. Commun. Rev.*, 32(3):75–88, 2002.

[89] W. John and S. Tafvelin. Analysis of internet backbone traffic and header anomalies observed. In *Proceedings of the 7th ACM SIGCOMM conference on Internet measurement*, pages 111–116. ACM, 2007.

[90] Thomas Karagiannis, Andre Broido, Nevil Brownlee, kc claffy, and Michalis Faloutsos. Is P2P dying or just hiding? In *Global Telecommunications Conference, 2004. GLOBECOM'04. IEEE*, volume 3, pages 1532–1538. IEEE, 2004.

[91] Thomas Karagiannis, Andre Broido, Michalis Faloutsos, and Kc claffy. Transport layer identification of P2P traffic. In *IMC '04: Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*, pages 121–134, New York, NY, USA, 2004. ACM Press.

[92] Alan Kay, Julian Lombardi, Mark P. McCahill, Andreas Raab, David P. Reed, and David A. Smith. Croquet. `http://www.opencroquet.org/`, 2007. [Online; accessed 29-September-2010].

[93] T. Kelly. Scalable TCP: Improving performance in highspeed wide area networks. *ACM SIGCOMM Computer Communication Review*, 33(2):91, 2003.

[94] Brian W Kernighan and Dennis Ritchie. *The C Programming Language*. Prentice Hall, 2nd edition, 1988.

[95] Ken Keys, David Moore, Ryan Koga, Edouard Lagache, Michael Tesch, and k claffy. The architecture of CoralReef: an Internet traffic monitoring software suite. In *PAM2001 — A workshop on Passive and Active Measurements*. CAIDA, RIPE NCC, April 2001. `http://www.caida.org/tools/measurement/coralreef/` [Online; accessed 22-October-2010].

[96] Jaecheol Kim, Jaeyoung Choi, Dukhyun Chang, Taekyoung Kwon, Yanghee Choi, and Eungsu Yuk. Traffic characteristics of a massively multi-player online role playing game. In *NetGames '05: Proceedings of 4th ACM SIGCOMM workshop on Network and system support for games*, pages 1–8, New York, NY, USA, 2005. ACM.

[97] James Kinicki and Mark Claypool. Traffic Analysis of Avatars in Second Life. In *Proceedings of the 18th ACM International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV)*, Braunschweig, Germany, May 2008.

[98] J. Kirriemuir. Early summer 2009 Virtual World Watch snapshot of virtual world activity in UK HE and FE. *Retrieved August http://content.imamu.edu.sa/Scholars/it/net/snapshot-six.pdf*, 24:2009, 2009.

[99] E. Kohler, M. Handley, and S. Floyd. Datagram Congestion Control Protocol (DCCP). RFC 4340 (Proposed Standard), March 2006.

[100] Eddie Kohler, Mark Handley, and Sally Floyd. Designing dccp: congestion control without reliability. *SIGCOMM Comput. Commun. Rev.*, 36(4):27–38, 2006.

[101] R. Krishnakumar. Kernel Korner: Kprobes-a Kernel Debugger. *Linux Journal*, (133):11, 2005.

[102] Sanjeev Kumar, Jatin Chhugani, Changkyu Kim, Daehyun Kim, Anthony Nguyen, Pradeep Dubey, Christian Bienia, and Youngmin Kim. Second life and the new generation of virtual worlds. *Computer*, 41(9):46–53, 2008.

[103] Chi-Anh La and Pietro Michiardi. Characterizing user mobility in second life. In *WOSP '08: Proceedings of the first workshop on Online social networks*, pages 79–84, New York, NY, USA, 2008. ACM.

[104] T. V. Lakshman and Upamanyu Madhow. The performance of tcp/ip for networks with high bandwidth-delay products and random loss. *IEEE/ACM Trans. Netw.*, 5(3):336–350, 1997.

[105] V. Lally, E.H. Magill, J. Magill, B. Canavan, M. Kolberg, M. Pomerantz, M. Sclater, J. Trinder, and J. & S. Brindley. Inter-life: Interoperability and transition. In *CAL09: Learning in Digital Worlds*, March 23-25th 2009.

[106] Tanja Lang, Philip Branch, and Grenville Armitage. A synthetic traffic model for quake3. In *ACE '04: Proceedings of the 2004 ACM SIGCHI International Conference on Advances in computer entertainment technology*, pages 233–238, New York, NY, USA, 2004. ACM.

[107] latifer, nebadon2025, j.veenvan, coyle@knifejaw.com, mcortez, bhardly, and james.stallings. Open metaverse foundation. `http://code.google.com/p/openmetaverse/`. [Online; accessed 15-November-2010].

[108] M. Laubach and J. Halpern. Classical IP and ARP over ATM. RFC 2225 (Proposed Standard), April 1998.

[109] Huiguang Liang, Mehul Motani, and Wei Tsang Ooi. Textures in second life: Measurement and analysis. In *ICPADS '08: Proceedings of the 2008 14th IEEE International Conference on Parallel and Distributed Systems*, pages 823–828, Washington, DC, USA, 2008. IEEE Computer Society.

[110] P. Lincroft. The Internet sucks: Or, what I learned coding X-Wing vs. TIE Fighter. *Gamasutra, Sep*, 3, 1999.

[111] Linden Labs. Second life havoc 4. `http://wiki.secondlife.com/wiki/Havok_4_Beta_Home`, November 2008. [Online; accessed 10-November-2010].

[112] Linden Labs. Second life client source (version 1.23.4). `http://secondlife.com/developers/opensource/downloads/2009/06/slviewer-src-viewer-1.23.4-r124025.tar.gz`, 15 June 2009. [Online; accessed 10-November-2010].

[113] Inc. Linden Research. Second Life Documentation. `http://wiki.secondlife.com/wiki/Documentation`. [Online; accessed 04-April-2011].

[114] Inc. Linden Research. Second life games. `http://secondlife.com/destinations/games`. [Online; accessed 28-March-2011].

[115] Linden Research, Inc. Second Life. `http://www.secondlife.com`, 23 June 2003. [Online; accessed 29-September-2010].

[116] Tim Lindholm and Frank Yellin. *The Java Virtual Machine Specification*. Prentice Hall, New York, NY, USA, 2nd edition, 1999.

[117] Makena Technologies. There. `http://www.there.com/`, 9 January 2003. [Online; accessed 29-September-2010].

[118] Matthew Mathis, Jeffrey Semke, Jamshid Mahdavi, and Teunis Ott. The macroscopic behavior of the TCP Congestion Avoidance algorithm. *SIGCOMM Comput. Commun. Rev.*, 27(3):67–82, 1997.

[119] Steven McCanne and Van Jacobson. The BSD Packet Filter: A New Architecture for User-level Packet Capture. In *USENIX Winter*, pages 259–270, 1993.

[120] R.M. Metcalfe and D.R. Boggs. Ethernet: Distributed packet switching for local computer networks. *Communications of the ACM*, 19(7):395–404, 1976.

[121] Microsoft. .NET Framework. `http://msdn.microsoft.com/netframework`, 13 February 2002. [Online; accessed 08-November-2010].

[122] J. Mogul. Efficient use of workstations for passive monitoring of local area networks. In *SIGCOMM '90: Proceedings of the ACM symposium on Communications architectures & protocols*, pages 253–263, New York, NY, USA, 1990. ACM Press.

[123] J. Mogul, R. Rashid, and M. Accetta. The packer filter: an efficient mechanism for user-level network code. In *SOSP '87: Proceedings of the eleventh ACM Symposium on Operating systems principles*, pages 39–51, New York, NY, USA, 1987. ACM Press.

[124] J.C. Mogul and J. Postel. Internet Standard Subnetting Procedure. RFC 950 (Standard), August 1985.

[125] Monoxide and tjc. Halo map tools v3.0. `http://hce.halomaps.org/index.cfm?fid=725`, 29 December 2004. [Online; accessed 08-October-2010].

[126] Andrew Moore, Christian Kreibich, James Hall, Euan Harris, and Ian Pratt. Architecture of a Network Monitor. In *In Passive & Active Measurement Workshop 2003 (PAM2003)*, 2003.

[127] Andrew W. Moore, Rolf Neugebauer, James Hall, and Ian Pratt. Network Monitoring with Nprobe. `http://www.cl.cam.ac.uk/users/iap10/nprobe2002.pdf`, 2002.

[128] moove. moove online. `http://www.moove.com/info_moove_online.htm`, September 2001. [Online; accessed 15-November-2010].

[129] Chip Morningstar and F. Randall Farmer. The lessons of Lucasfilm's Habitat. *Cyberspace: first steps*, pages 273–302, 1991.

[130] Chip Morningstar, Randy Farmer, and many others. Habitat, 1986.

[131] MW, Adam Frisby, MingChen, lbsa71, Sean Dague, Tedd, justincc, Teravus, Johan Berntsson, Charles Krinke, Dr Scofield, dahlia, mikem, Melanie Thielker, Homer Horwitz, Ian Brown, Crista Lopes, nlin, and Arthur Rodrigo S Valadares. OpenSimulator. `http://opensimulator.org`. [Online; accessed 29-September-2010].

[132] MW, Adam Frisby, MingChen, lbsa71, Sean Dague, Tedd, justincc, Teravus, Johan Berntsson, Charles Krinke, Dr Scofield, dahlia, mikem, Melanie Thielker, Homer Horwitz, Ian Brown, Crista Lopes, nlin, and Arthur Rodrigo S Valadares. Opensim source (version 0.6.7). `http://dist.opensimulator.org/opensim-0.6.7-release.tar.gz`, 14 October 2009. [Online; accessed 10-November-2010].

[133] J. Nagle. Congestion control in IP/TCP internetworks. RFC 896, January 1984.

[134] J. Nagle. On packet switches with infinite storage. RFC 970, December 1985.

[135] J. Neider, T. Davis, and M. Woo. *OpenGL. Programming guide*. Addison-Wesley Reading, MA, 1997.

[136] James Nichols and Mark Claypool. The effects of latency on online madden NFL football. In *NOSSDAV '04: Proceedings of the 14th international workshop on Network and operating systems support for digital audio and video*, pages 146–151, New York, NY, USA, 2004. ACM.

[137] K. Nichols, S. Blake, F. Baker, and D. Black. Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers. RFC 2474 (Proposed Standard), December 1998. Updated by RFCs 3168, 3260.

[138] Geoffrey Norris. For a new concert experience, get a second life, 15 September 2007.

[139] Novell and the Mono community. Mono. `http://www.mono-project.com/Main_Page`, 30 June 2004. [Online; accessed 08-November-2010].

[140] Manuel Oliveira and Tristan Henderson. What online gamers really think of the Internet? In *NetGames '03: Proceedings of the 2nd workshop on Network and system support for games*, pages 185–193, New York, NY, USA, 2003. ACM.

[141] Iain Oliver, Colin Allison, and Alan Miller. Traffic Management for Multi-User Virtual Environments. In *The 10th Annual PostGraduate Symposium on The Convergence of Telecommunications, Networking and Broadcasting (PGNET 2009)*, 2009.

[142] Iain Oliver, Colin Allison, and Alan Miller. Mongoose: A TCP Fair Second Life client. In *Proceedings of the 11th Annual PostGraduate Symposium on The Convergence of Telecommunications, Networking and Broadcasting (PGNet 2010)*, 2010.

[143] Iain Oliver, Alan Miller, and Colin Allison. tcpread: An aid to analysing tcpdump traces. In *Proceedings of The 8th Annual Postgraduate Symposium on The Convergence of Telecommunications, Networking and Broadcasting (PGNET 2007)*, 2007.

[144] Iain Oliver, Alan Miller, and Colin Allison. A User Space Implementation of DCCP. In *Proceedings of The 9th Annual Postgraduate Symposium on The Convergence of Telecommunications, Networking and Broadcasting (PGNET 2008)*, 2008.

[145] Iain A. Oliver, Alan H.D. Miller, and Colin Allison. Virtual worlds, real traffic: interaction and adaptation. In *Proceedings of the first annual ACM SIGMM conference on Multimedia systems*, MMSys '10, pages 305–316, New York, NY, USA, 2010. ACM.

[146] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose. Modeling TCP Throughput: A Simple Model and its Empirical Validation. In *ACM SIGCOMM Computer Communication Review*, volume 28, pages 303–314, Amherst, MA, USA, 1998. ACM, University of Massachusetts.

[147] C. Partridge and R.M. Hinden. Version 2 of the Reliable Data Protocol (RDP). RFC 1151 (Experimental), April 1990.

[148] V. Paxson and M. Allman. Computing TCP's Retransmission Timer. RFC 2988 (Proposed Standard), November 2000.

[149] Vern Paxson. Bro: a system for detecting network intruders in real-time. *Comput. Networks*, 31(23-24):2435–2463, 1999.

[150] C. Perkins. IP Encapsulation within IP. RFC 2003 (Proposed Standard), October 1996.

[151] Colin Perkins and Ladan Gharai. RTP and the Datagram Congestion Control Protocol. In *ICME*, pages 1521–1524. IEEE, 2006.

[152] P. Phaal, S. Panchen, and N. McKee. InMon Corporation's sFlow: A Method for Monitoring Traffic in Switched and Routed Networks. RFC 3176 (Informational), September 2001.

[153] Phoenix Linden. Second life: Packet layout. `http://wiki.secondlife.com/wiki/Packet_Layout`, 31 October 2006. [Online; accessed 10-November-2010].

[154] J. Postel. Telnet Protocols. (318), April 1972. Updated by RFC 435.

[155] J. Postel. DoD standard Internet Protocol. RFC 760, January 1980. Obsoleted by RFC 791, updated by RFC 777.

[156] J. Postel. User Datagram Protocol. RFC 768 (Standard), August 1980.

[157] J. Postel. Internet Control Message Protocol. RFC 777, April 1981. Obsoleted by RFC 792.

[158] J. Postel. Internet Control Message Protocol. RFC 792 (Standard), September 1981. Updated by RFCs 950, 4884.

[159] J. Postel. Internet Protocol. RFC 791 (Standard), September 1981. Updated by RFC 1349.

[160] J. Postel. Transmission Control Protocol. RFC 793 (Standard), September 1981. Updated by RFC 3168.

[161] J. Postel. Standard for the transmission of IP datagrams over experimental Ethernet networks. RFC 895 (Standard), April 1984.

[162] P. Pradhan, S. Kandula, W. Xu, A. Shaikh, and E. Nahum. Daytona: A user-level TCP stack. *Available HTTP: http://nms.lcs.mit.edu/~kandula/data/daytona.pdf*, 2002.

[163] K. Ramakrishnan, S. Floyd, and D. Black. The Addition of Explicit Congestion Notification (ECN) to IP. RFC 3168 (Proposed Standard), September 2001.

[164] K. K. Ramakrishnan and R. Jain. A binary feedback scheme for congestion avoidance in computer networks. *ACM Trans. Comput. Syst.*, 8(2):158–181, 1988.

[165] K. K. Ramakrishnan and Raj Jain. A binary feedback scheme for congestion avoidance in computer networks with a connectionless network layer. *SIGCOMM Comput. Commun. Rev.*, 18:303–313, August 1988.

[166] E. Rescorla. HTTP Over TLS. RFC 2818 (Informational), May 2000.

[167] J.K. Reynolds and J. Postel. Assigned numbers. RFC 870 (Historic), October 1983. Obsoleted by RFC 900.

[168] J.K. Reynolds and J. Postel. Assigned Numbers. RFC 900 (Historic), June 1984. Obsoleted by RFC 923.

[169] I. Rhee and L. Xu. CUBIC: A new TCP-friendly high-speed TCP variant. In *Proc. PFLDnet*, volume 2005. Citeseer, 2005.

[170] Alan Ruddle, Colin Allison, and Ross Nicoll. Analysing the latency of world wide web applications. *Softw., Pract. Exper.*, 33(14):1301–1322, 2003.

[171] J. H. Saltzer, D. P. Reed, and D. D. Clark. End-to-end arguments in system design. *ACM Trans. Comput. Syst.*, 2(4):277–288, 1984.

[172] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. RTP: A Transport Protocol for Real-Time Applications. RFC 3550 (Standard), July 2003.

[173] H. Schulzrinne, A. Rao, and R. Lanphier. Real Time Streaming Protocol (RTSP). RFC 2326 (Proposed Standard), April 1998.

[174] Nathan Sheldon, Eric Girard, Seth Borg, Mark Claypool, and Emmanuel Agu. The effect of latency on user performance in Warcraft III. In *NetGames '03: Proceedings of the 2nd workshop on Network and system support for games*, pages 3–14, New York, NY, USA, 2003. ACM.

[175] W. Simpson. The Point-to-Point Protocol (PPP). RFC 1661 (Standard), July 1994. Updated by RFC 2153.

[176] David A. Smith, Alan Kay, Andreas Raab, and David P. Reed. Croquet - a collaboration system architecture. *Proceedings of the First Conference on Creating, Connecting and Collaborating Through Computing (C5'03)*, pages 2–9, 2003.

[177] IL Smith. Joint Academic NETwork (JANET). *Computer Networks and ISDN Systems*, 16(1-2):101–105, 1988.

[178] Joel Sommers, Paul Barford, Nick Duffield, and Amos Ron. Improving accuracy in end-to-end packet loss measurement. In *SIGCOMM '05: Proceedings of the 2005 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 157–168, New York, NY, USA, 2005. ACM Press.

[179] R. Stewart. Stream Control Transmission Protocol. RFC 4960 (Proposed Standard), September 2007.

[180] D. Stødle. ptunnel – Ping Tunnel. 2005.

[181] Bjarne Stroustrup. *The C++ Programming Language*. Addison-Wesley, 1986.

[182] T Sturgeon, A Miller, and C Allison. A wifi virtual laboratory. In *Information and Computer Sciences, The Higher Education Academy, Dublin*, pages 207–211, 2006.

[183] Thomas Sturgeon, Colin Allison, and Alan Miller. 802.11 wireless experiments in a virtual world. *ACM SIGCSE Bull.*, 41(3):85–89, 2009.

[184] Thomas Sturgeon, Colin Allison, and Alan Miller. 802.11 wireless experiments in a virtual world. *SIGCSE Bull.*, 41:85–89, July 2009.

[185] Mirko Suznjevic, Maja Matijasevic, and Ognjen Dobrijevic. Action specific Massive Multiplayer Online Role Playing Games traffic analysis: case study of World of Warcraft. In *NetGames '08: Proceedings of the 7th ACM SIGCOMM Workshop on Network and System Support for Games*, pages 106–107, New York, NY, USA, 2008. ACM.

[186] Philipp Svoboda, Wolfgang Karner, and Markus Rupp. Traffic analysis and modeling for world of warcraft a mmog. In *ICC 2007*. ICC 2007, IEEE International Communication Conference (ICC), Glasgow, Irland, 2007.

[187] Rebecca Sweetman and Evi Katsara. The acropolis basilica project, sparta: A preliminary report for the 2000 season. Technical report, British School at Athens, 2002.

[188] Matteo Varvello, Fabio Picconi, Christophe Diot, and Ernst Biersack. Is There Life in Second Life? In *Proceedings of ACM CoNEXT 2008*, December 2008.

[189] Wikipedia. List of TCP and UDP port numbers — Wikipedia, The Free Encyclopedia, 2007. [Online; accessed 25-April-2007].

[190] L. Xu, K. Harfoush, and I. Rhee. Binary increase congestion control (BIC) for fast long-distance networks. In *IEEE INFOCOM*, volume 4, pages 2514–2524. Citeseer, 2004.

[191] A. Yarali and A. Cherry. Internet Protocol Television (IPTV). pages 1 –6, nov. 2005.

[192] Masanobu Yuhara, Brian N. Bershad, Chris Maeda, and J. Eliot B. Moss. Efficient packet demultiplexing for multiple endpoints and largemessages. In *WTEC'94: Proceedings of the USENIX Winter 1994 Technical Conference on USENIX Winter 1994 Technical Conference*, pages 13–13, Berkeley, CA, USA, 1994. USENIX Association.

Appendix A

Paper

We have written the following papers which have been published relating to this work:

A User Space Implementation of DCCP [144].

Virtual worlds, real traffic: interaction and adaptation [145].

Metaverses as a Platform for Game Based Learning [65].

Mongoose: A TCP Fair Second Life client [142].

Traffic Management for Multi-User Virtual Environments [141].

Traffic Management for Multi-User Virtual Environments [143].

Appendix B

Server Side traffic management

A system which uses a modified client to produce a Transmission Control Protocol (TCP) fair bandwidth utilization rate was created. This allowed the rate calculation to be changed but has not allowed to significant change to be made to the bandwidth and priority given to different types of traffic to be changed. A modified server and client would allow the channels to be changed and for the congestion control to be performed at the server.

Section B.1: The design of the implementation of the system is described.

Section B.2: The implementation details of the system are described.

Section B.3: The evaluation of the User-Space DCCP Library (USDL) is described.

B.1 OpenSim over DCCP server and client

Separating out the data and control of that data can be done in a number of different ways. It could be done with separate UDP packets between the same hosts, using the same or different port numbers. It could also be achieved by adding an extra header to the packets transported using UDP. Another option is to use a different transport layer protocol. Second Life doesn't require reliability or ordering for its packets as it can deal with loss. Second Life circuits are connection like so there is no requirement that the protocol not be connection oriented. The transport level protocol DCCP has these properties, it is also designed to be able to incorporate different congestion control systems and to detect loss in different ways.

The circuits will use DCCP connections to transport packets. The calculation of RTT and detection of loss can be performed in the implementation of DCCP. DCCP can include timing information in any of the reply packets and so can do its RTT calculations without requiring specialized packets. The USDL library described in appendix C will be used as the implementation of DCCP. Its congestion control systems of USDL can be altered to report information about the state of the network to gain more information. This information about the network conditions can then be communicated to the controller. This allocation will be communicated to the application to allow it to set up the throttles to the different channels to guarantee the bandwidth avatar traffic. This will allow it provide different quality of service to the different types of packet. The system will be able to allocate resources to the connections to meet its needs and maximize the utility to the users. The overall traffic produced by the system should not exceed its fair allocation. Figure B.1 shows the structure of the proposed system.

C# language [46] bindings were created for USDL. This has then been integrated into OpenSim so that it uses DCCP to send and receive data. The library will then be modified to report more information to OpenSim.

The DCCP library and OpenSim server communicate through C# language bindings and wrapper classes. The library handles controlling sending packets and controlling the rate at which it sends packets. The library gathers information about the congestion state of the network. This information is used by a congestion control system to determine the correct packet sending rate. The congestion information along with the values calculated by congestion control systems are available to OpenSim from the library through the wrapper library. This information will be entered into the existing system which controls packet transmission. This will allow the sending rate to meet the users settings whilst not exceeding its fair allocation of bandwidth.

The library will be able to provide more information than the available bandwidth, including information about packet loss and timing information. This information is used to determine the type and volume of information that

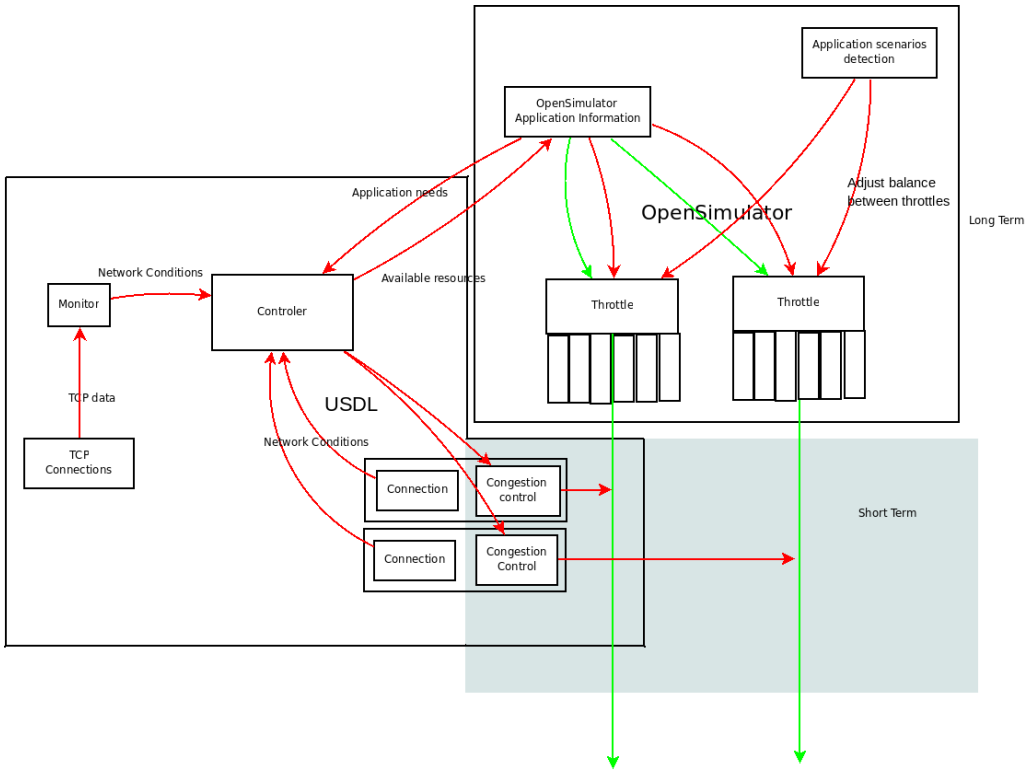


Figure B.1: The overall structure of the system.

should be sent to the clients to maximise the utility of the users. The detail of the information sent to the client will be reduced, when the amount of available bandwidth is below a threshold.

The client will be modified in two ways. First the client will be adapted to use the library to send and receive DCCP packets. Secondly the client will be modified to allow it to detect the correct sending rate for the server from the packet loss detected at the client end. This information will be sent to the server using the existing system to allow the user to control the data transfer speed. The library will be modified to allow it to be used to provide this type of traffic management to traffic not using DCCP for transport. The client will inform the library when packets are lost and this information will be fed into a congestion control system to give the client a value for its fair share of the available bandwidth that it can use in its calculations and send to the server.

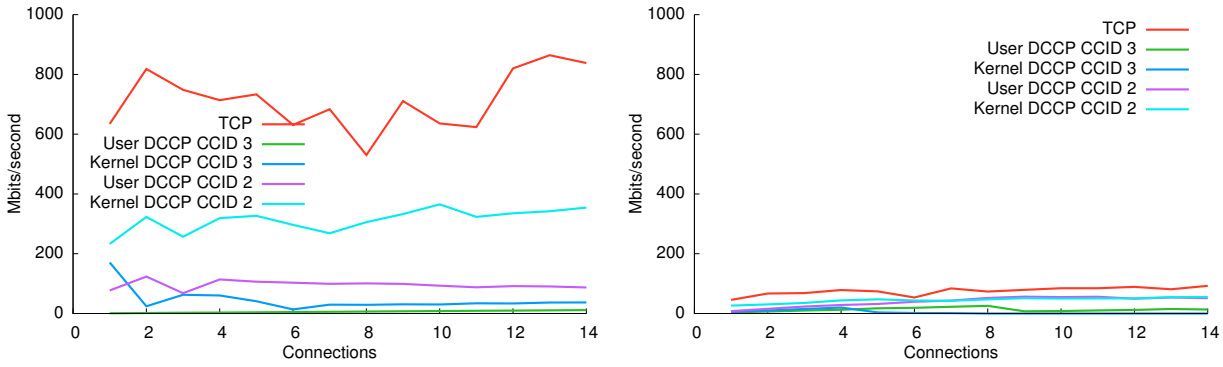
The existing congestion control systems in the library will be altered to put the information that they have into a new system to store this information and make it available to user programs and other congestion control systems. A monitor will be implemented to allow the system to track the state of other connections not under its control and so improve its knowledge of the congestion state of the network.

The other commonly used transport protocol UDP does not provide reliability, so it is used by applications that can deal with packet loss. The problem with UDP is that it does not provide any congestion control system, so an application can produce as many packets as it wants, using up all of the available bandwidth. DCCP was created to provide a system similar to UDP in that it is unreliable but that includes a congestion control mechanism.

B.2 Client and server communicating over DCCP

The Second life client

To use USDL and so DCCP packets for transporting it packets the Second Life client was altered. The standard Second Life client uses one UDP socket to send and receive packets with clients, this is possible because UDP is not connection orientated. As DCCP is connection oriented it is necessary to have a separate socket for each circuit. The `llcircuit` class that hold information about each circuit, as described in section 6.3, this class hold the information about the circuit including the sequence number information and RTT information. The `llcircuit` data structure does not hold a file descriptor for the UDP socket as there is only one so there would be propose. The `llcircuit`



(a) RTT of 10 ms

(b) RTT of 100 ms

Figure B.2: The bandwidth utilisation against number of connections for the two DCCP implementations and both congestion control systems and TCP

data structure was altered by adding a file descriptor for the DCCP connection that the circuit uses. The `llcircuit` constructor creates and connections a DCCP socket and stores the FD and it is deconstructor closes the socket.

The actual opening and closing of DCCP sockets is performed by new function. The socket creation function first creates a new DCCP socket then puts it into non blocking mode and sets the send a receive buffer sizes. The service code is then set and the socket is connected to the server. The FD is stored in a map from the open sockets to the host they connect to, the FD is also stored in the `llcircuit` data structure. The function to close the connection retrieves the host, using the socket's descriptor, from the dictionary and closes the socket and removes its FD from the dictionary. The design of the standard Second Life client has one function retrieves packets from the socket for all of the circuits, this design was retained in the modified client. The modified client has a function that uses poll to check for incoming packets from the connections. The new receive function adds the FDs to an array for the poll function. The function then iterates through the list of socket and reads a packet from one of the socket. A new packet sending function has been created which takes the socket to use to send the packet.

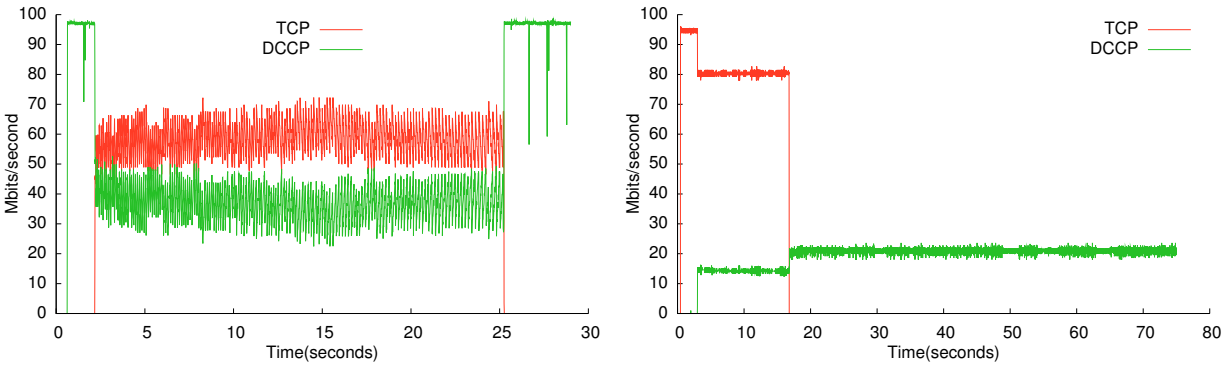
The `llpacketbuffer` class has a constructor that reads a packet from the network, this was altered to use the new function. The only other class the calls the functions in net is `llpacketring` that already gets passed a file descriptor for the socket to be used to send packets. When the reliable packets are created they contain a file descriptor this is now provided by the circuit, when unreliable packets are sent the `sendPacket` function is called with the FD from the circuit.

OpenSim server

The OpenSim server is implemented in C# because of this it cannot directly call the C API for USDL. A C# library was created to provide access to the library from the C#. The USDL library provides an API for the functions specific to sockets, the other functions performed on the FD of the socket is performed by the standard functions provided by the operating system.

USDLsocket The Socket class in the `USDLsocket` package is designed to be very similar to the socket API provided by mono. It calls the functions in the USDL library to create and manipulate a DCCP connection. A small library was created to construct the native data structure that are used to change settings. The `syscall` class is used to call the systems calls that are used when using a FD. A number of other classes were also created. A `SocketException` class to map the errors returned by the Unix system calls to the error codes used by mono. A number of functions were created to pack and unpack the data structures that are used for the calling USDL functions.

A class was also created that creates a socket to accept incoming connections and will receive a packet from any of the accepted sockets, this class is called `DCCPserver`. This allows the OpenSim server to maintain the design were a single function reads in all of the packets. This allows the library to be more quickly integrated into existing systems that use UDP. The connections can then be retrieved from this class.



(a) DCCP CCID 2 and a TCP connection

(b) DCCP CCID 3 and a TCP connection

Figure B.3: The number of bytes per second for one DCCP connections and one TCP connection, sampled once every 10 milliseconds, on a network with no routers.

OpenSim Only a small number of classes in the OpenSim server access the UDP socket directly. These references were replaced with references a DCCPserver object that was given a service code and set listening. This was sufficient to create a server capable of using DCCP for its communication with the client.

B.3 Evaluation of TCP fair algorithms using DCCP

USDL has been evaluated to determine that it correctly works and is able to provide the functionality necessary for the OpenSim server. The library should be able to operate correctly when transporting different types of traffic including application where there is less data to send than can be sent, such as OpenSim.

A network was created with three computers connected to a 100 Mbit switch. With two computers sending packets and one receiving them. A number of programs were created to test USDL.

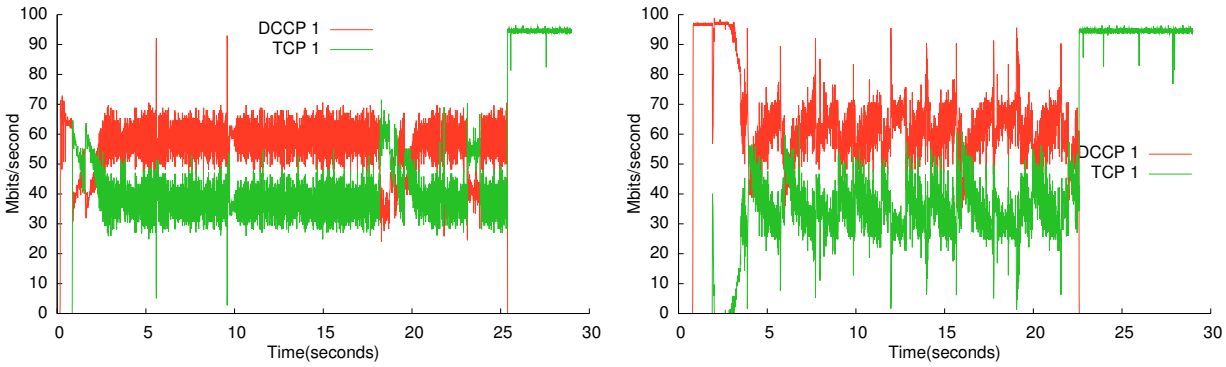
The library has been tested using a number of programs. Connections were established and data was sent, the speed of the data transmission and the loss rate were compared to the Linux kernel implementation, packet trace were also examined to ensure that packet level interaction of a single connection was correct and that the congestion control systems did limit the transmission of packets. A program was created to accept a large number of incoming connection from another program, these were tested using the library and the Linux kernel implementation of DCCP to detect problems with large numbers of connections. The feature negotiation of the library was tested by changing the congestion control preferences in programs to ensure that the negotiation was performed correctly, in doing this it was discovered that Linux kernel implementation did not correctly implement this.

The bandwidth utilisation against number of connections for the two DCCP implementations and both congestion control systems and TCP is shown in figure B.2, this was recorded on a 1 Gbit connection. The x-axis shows the number connections and the y-axis shows bandwidth utilization in Mbps. The red line is TCP connections, the green line is USDL's CCID 3 connections, the blue line is the Linux kernel's CCID 3 connections, the purple line is USDL's CCID 2 and the light blue line is the Linux kernel's CCID 2 connections.

(a) The graph for a RTT of 10 ms is shows in figure B.2a. At low RTTs increasing the number of connections does not necessarily increase the amount of bandwidth used. The CCID 3 implementations do not work well with a large number of connections.

(b) The graph for a RTT of 100 ms is shown in figure B.2b. The different types of connections are closer together in their bandwidth utilization when the RTT is larger. The kernel CCID 3 does better than the USDL one with a small number of connections, but over 4 connections it doesn't work as well.

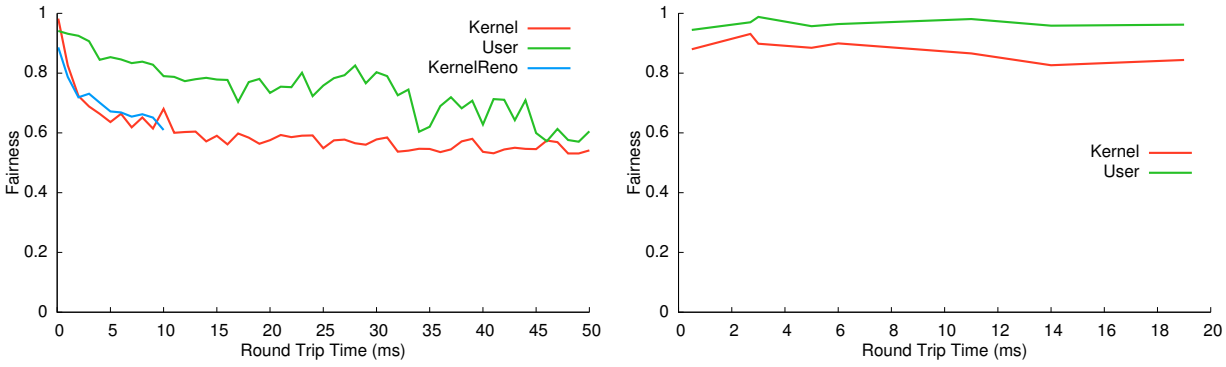
They show that user level CCID 2 is able to operate at around 200 Mbps. These graphs show that TCP performs better than DCCP at normally encountered round trip times, however when the round trip time increases sufficiently the performance of TCP decreases. The performance of CCID 3 when only competing against itself is better than that of CCID 2, when it competes against itself.



(a) User Level CCID 2 and a TCP connection

(b) Kernel Level CCID 2 and a TCP connection

Figure B.4: The number of bytes per second for one DCCP connections and one TCP connection, sampled once every 10 milliseconds, on a network with 1 router.



(a) Without a router

(b) With a router

Figure B.5: The fairness between a TCP connection and a DCCP connection against the RTT of the link.

Both of the congestion control system supported by DCCP are shown competing against a TCP connection in figure B.3.

(a) The TCP-like congestion control is show in figure B.3a, it shows that the TCP like congestion control system is able to fairly share bandwidth with a TCP connection. The user DCCP connections will not use more than there fair share of the bandwidth, however two connections never exactly share the same amount of bandwidth. User DCCP when competing against TCP in this setup always used less bandwidth than TCP when the two compete for bandwidth.

(b) The DCCP's TCP fair rate control connection competing with a TCP connection, is shown in figure B.3b, is having difficulty obtaining its fair share of bandwidth when competing with TCP connections.

The TCP-like congestion control system provides congestion control that is similar to that provided by TCP. It reacts quickly to changes in the network conditions and is TCP fair. It uses the most bandwidth possible whilst staying TCP fair. The rate based TCP friendly rate control (TFRC), congestion control system is intended to provide a steady rate that is TCP fair. However in this experiment it can be seen that the current version is unable to get its TCP fair share of the available bandwidth.

Graphs of DCCP and TCP connections on a network containing one router are shown in figure B.4. This was on a network with one router, with a DCCP connection and a TCP connection competing for bandwidth. The x-axis shows the time in seconds since the beginning of the trace. The y-axis shows the bandwidth utilization in Mbps. The red line shows the bandwidth for the DCCP connection. The green line shows the bandwidth for the TCP connection.

(a) A USDL DCCP connection using CCID 2 competing against a TCP connection is show in figure B.4a. The two connections have similar amounts of bandwidth utilization, though they are not the same. The amount of

bandwidth used by the connections crosses over but goes back to the arrangement were the DCCP connection is using more bandwidth.

(b) A Linux kernel DCCP connection using CCID 2 competing against a TCP connection is shown in figure B.4b. The DCCP connection is using more bandwidth in this trace. The transfer rate is less stable than the transfer rate in figure B.4a. The difference between the bandwidth used by the connections is larger.

On this network DCCP CCID 2 and TCP are also able to fairly share bandwidth.

DCCP and TCP react differently to changes in the round trip time, as a result the fairness between a TCP and a DCCP connection on the same link will be different depending on the RTT of that link. Figure B.5a shows the fairness [86] between a TCP connection and a DCCP connection against the RTT of the link. In all of the cases the TCP connection takes more of the bandwidth than the DCCP connection. At a very short RTT the kernel implementation is able to achieve greater fairness than user space implementation, however when the RTT is greater than 1 ms the user space implementation achieves better fairness than the kernel implementation. The method of measuring fairness gives a value between 1/n and 1 where n is the number of connections, in this case 2. 1 would indicate a completely fair sharing of the available bandwidth and 0.5 would indicate that one of the connections is taking up all of the available bandwidth. The TCP-like congestion control system stays within its TCP fair allocation of bandwidth.

Figure B.5b shows the mean fairness between DCCP CCID 2 connection and a TCP connection for both user and the kernel DCCP implementations, against the RTT. The router introduces greater delay into the network so it is impossible to get fairness values for a RTT less than 1 ms. For RTT greater than 1 ms connections achieve better fairness with a router than without. The presence of a router is assumed in the design of congestion control systems that attempt to compete fairly with TCP. Further the routers are assumed to use Random early drop (RED) to deal with too many packets being received.

B.4 Future Work

The future work is to evaluate this modified server under different network conditions.

B.5 Conclusion

A traffic management system using a modified client and server has been designed to be TCP fair. This system uses a modified channel system which protects the avatar traffic from the other traffic. This system uses the USDL implementation of DCCP to provide the congestion control system, with a modified congestion control system that gives more information to the server. The channel throttles are set depending on the amount of bandwidth to use to provide differential QoS to the different types of packets.

The design is to create C# bindings for the USDL library to integrate it into the OpenSim server. The client is modified to use the USDL library. The USDL library has been evaluated to determine that is able to handle sufficient traffic to be used for this propose.

A modified server where the throttle values are calculated at the server end of the connection would be able to adapt more quickly and would have less problems with a misbehaving client. A system has been designed that uses an implementation of DCCP to provide the throttle values. This designed system has been implemented and is able to function. The DCCP implementation has been tested and is able to handle the required amount of bandwidth.

Appendix C

User Level DCCP

The Internet provides an unreliable service for transferring packets of data. When the number of packets sent onto the network exceeds its ability to transport those packets the network becomes congested and packets can be lost. The introduction of reliability at the transport layer meant that when the network was most congested the amount of load on the network would increase until very little or no traffic across the network. In 1986 Jain [85] described a system to detect congestion on a network and prevent congestion collapse, with this system when a packet is transmitted a timer is started, if the packet has not acknowledged within an amount of time the packet is considered to have been lost, this is taken to indicate that network is congested. A variable called the window size (WS) limits the number of packets that are in flight in the network at anyone time. When a packet is lost the WS is reduced to one. For every N successful packet transmissions the WS is increased by one, up to the window specified by the other direction. This paper however advocates the use of a congestion control system where the window size is reduced to one when a packet is lost, because when packet loss occurs the number of network and router queues are all full and will take some time to empty. In 1988 Jacobson described a congestion control system designed to reach a steady state of packet send, for networks without changing conditions. To get to this state the connection starts in the slow start phase were the window starts at one packet and increases by one for each acknowledgement received. When packets are lost a variable SSTHRESH is set to half the congestion window and the congestion window is set to one packet. The connection then enters slow-start, until the congestion window is greater-than or equal to SSTHRESH, the connection then enter the congestion avoidance stage. In the congestion avoidance stage the window size is increased by one over the window size for each acknowledgement received.

TCP provides reliable data transfer and has congestion control implemented on top of it. This provides the service required by a large number of the applications that the Internet is used for and by providing congestion control it prevents congestion collapse on the Internet. TCP provides reliable in order and byte stream oriented data delivery, it is good for applications that require to transfer large amounts of data without time constraints. As it provides in order delivery if data is lost it must be re-sent before any data after it is available to the application. For applications where latency is important, and reliability is not essential UDP [156] has been commonly used. UDP does not provide reliability or congestion control, so packets that are lost are not retransmitted and data is delivered to the application out-of-order and with gaps, as it arrives. This allows the applications to meet their timing constraints however it doesn't prevent congestion on the network. To avoid congestion such applications employ a number of techniques. Some systems limit their traffic to set level or attempt to reduce the amount of traffic that is to be sent over the network so that there is no need for congestion control. They may alternatively use a congestion control system at a level above the transport level.

Congestion control at the application level could be implemented using a system unique to the application, for media streaming there is an existing protocol, RTP in conjunction with RTSP, that provides sufficient feedback for the application to implement its own congestion control system. RTP provides payload type identification, sequence numbers, time-stamping and delivery monitoring. There is also a proposal to include TFRC in RTP [66].

An alternative to implementing congestion control at the application level is to implement congestion control at the transport layer. DCCP [99] is a relatively new protocol that provides congestion control without requiring that data transfer is reliable or in the correct order. In [55] the problem statement for DCCP is described. As well as providing congestion control DCCP is intended to work with network address translators (NAT) and firewalls, this is possible as it is a connection oriented protocol. There are a number of implementations though currently only the Linux kernel space version is complete. Creating user space libraries is less complicated than creating and editing

kernel space modules as crashes are less of a problem and there are more tools available. DCCP is designed to allow for a number of different congestion control mechanisms. To aid in the development and evaluation of new congestion control mechanisms it was decided to create an implementation of DCCP in user space. This implementation will allow DCCP to be used on operating systems that currently lack support for DCCP. It has so far been used on both Linux and FreeBSD without DCCP support installed and Linux DCCP with DCCP support.

The structure of the rest of this chapter will have the following structure:

Section C.1: DCCP will be described.

Section C.2: User space implementations are described.

Section C.3: The library is described.

Section C.4: The processes that happen in the library are described.

Section C.5: The evaluation of the library is described.

C.1 Datagram Congestion Control Protocol (DCCP)

DCCP is a transport level protocol that provides congestion control, without reliability. In [55] the problem statement for DCCP is described. As well as providing congestion control DCCP is intended to work with network address translators (NAT) and firewalls, this is possible as it is a connection oriented protocol. [99] is the RFC that describes DCCP.

RTP also includes sufficient feedback about packet loss that an application could use it to control its packet transmission rate. RTP, on its own, does not include a mechanism for sending feedback to the sending end of the connection. DCCP's purpose is to provide congestion control to applications so that they do not need to implement their own congestion control system. If the features provided by RTP are required there is a specification for RTP over DCCP [151].

RTP [172], in conjunction with RTSP [173], can provide information to the application about the state of the network. This information can be used for the application to implement its own congestion control system. RTP provides payload type identification, sequence numbers, time-stamping and delivery monitoring. This allows an application to react in the way that best suites their needs rather than rely on a congestion control system that does not meet the needs of the application. The application will however need to implement its own congestion control system, or use a library. DCCP provides congestion control to the application with very little effort from the application. RTP was originally intended for use in situations where it would not have to deal with congestion. It was therefore not designed to include congestion control systems. DCCP was designed to provide congestion control. The design of DCCP keeps the monitoring of packet loss and reacting to congestion out of the application and in the implementation of the protocol. The features that DCCP has to support congestion control were not designed for any specific type of congestion control, instead they were designed to be capable of supporting different types of congestion control. This allows new congestion control systems to be created for DCCP and used by applications without large or any changes to the application.

In [100] DCCP is described as well as the reasons for its creation. Currently applications that require packets to be delivered within certain time constraints or not at all have used UDP [156] as it allows packets to be delivered out-of-order and not at all, however it does not provide any type of congestion control so applications either implement their own or have no congestion control mechanism. Applications in userspace can also use an application level protocol such as RTP [172] to provide the necessary infrastructure to allow congestion control, possible with TCP friendly rate control [66]. The purpose of DCCP is to provide congestion control to applications that would otherwise use UDP, allowing them to get feedback about packet delivery without having to implement acknowledgement or congestion control systems. The requirements of different applications on a congestion control system are different, for this reason DCCP was designed so that new congestion control systems can be added to it.

DCCP includes support for Explicit Congestion Notification (ECN) [163] which allows routers to mark packets instead of dropping them to indicate that it is congested. The sender must then be informed that a marked packet was received. As packets are marked instead of being dropped the congestion control system should react as if marked packets had been dropped. In TCP [160] [45] the use of ECN must be negotiated before it is used, as ECN was created after TCP, so support cannot be assumed to exist. DCCP however was created after ECN, so the default is to assume

that both ends of a connection can inform the other end when marked packets arrive. Implementations of DCCP can however indicate that they are unable to access the ECN bits of packets.

Currently DCCP has two congestion control systems. The default is TCP-like congestion control [57] that provides a congestion control system similar to the one used by TCP as is described in [4]. This allows applications to receive as much bandwidth as possible, whilst having similar behaviour to TCP's congestion control. It also has many of the other properties of TCP, in that it favour's bandwidth over other quality of service parameters. It does not provide reliability and ordering so the application gets the packet as it arrives rather than having to wait for any more packets to arrive. This congestion control system is not intended to affect delay or jitter. It is intended to reduce loss due to congestion, whilst maximising available bandwidth so by design it does not minimise the variation in the available bandwidth, rather it rapidly changes the amount of available bandwidth.

DCCP includes service numbers that indicate the type of application level traffic is carried, if this is used correctly this will allow priority to be given to certain types of traffic without the need to perform traffic classification [13].

Currently DCCP has two congestion control systems. The default is TCP like congestion control [57] that provides a congestion control system similar to the one used by TCP, which is described in [4]. This allows applications to receive as much bandwidth as possible, whilst having similar behaviour to TCP's congestion control.

The other congestion control system is TCP-Friendly Rate Control (TFRC) for DCCP [58], which provides a smoother rate of packet transmissions whilst remaining TCP friendly, at the expense of not achieving the maximum possible speed. TFRC is described in [72].

The Linux implementation is intended to be integrated with a congestion manager [8], there also a plan to integrate the library with a congestion manager.

DCCP is able to deal with the amounts of bandwidth that MUVEs, in general use and Second Life in particular, use. It is designed to be able to support new congestion control systems so it will be possible to add a congestion control system that meets the particular needs of MUVEs. Its similarity to UDP will mean that it will take less alteration to change the existing systems to use UDP. With a user space implementation it will easier to develop a new congestion control system that has the desired properties works correctly. With a user space implementation it will be possible to more closely integrate the implementation with client and server, this will allow the interface to be changed to allow information to be exchanged between the application and the protocol implementation. It will also allow DCCP to be used on systems that do not currently support DCCP.

C.2 Userspace network protocol implementations

There are a number of user space implementations of network stacks mostly focusing on TCP. These stacks were created for different reasons, so they have different designs and structures to suit the purpose for which they were created.

Daytona [162] is a user space implementation of TCP. It uses libpcap to receive packets and raw sockets are used to send packets. In order to prevent the host operating system from reacting to TCP packets that are part of Daytona's connections firewall rules are created to cause the operating system to drops incoming packets for the ports that are being used by Daytona.

Alpine [47] is a library that allows the network stack of FreeBSD to be run in user space on any UNIX. This implementation is designed to simplify the process of experimenting with changes to the TCP stack by allowing the changes to be made without restarting the operating system. It provides to the network stack all of the services that would usually be provided by the kernel. A network stack requires a device over which it can send and receive packets to satisfy this requirement a fake network device was created. Packet transmission is achieved by using a raw socket. To receive packets libpcap is used to, in order to avoid getting packets intended for other applications a BPF is used.

Minet [44] is a user space TCP/IP stack designed to support computer networking courses at undergraduate and graduate level, allowing the students to implement modules to form part of a TCP/IP stack. This implementation is implemented in C++ and designed specifically for educational purposes. As a result of this it was not intended to match the performance of other implementations.

The paper [21] describes a user space implementation of TCP where there is also a kernel space component that performs the demultiplexing. The user-space library uses a new specialised socket interface to the kernel part.

There are a number of systems for network monitoring that track the state of connections in user space in order to gather statistics about the traffic and connection it contains. This can include a simulation of part of the behaviour of an end system.

There are also kernel space implementations of DCCP, the most complete is the Linux kernel implementation. This implementation is quite complete and usable though it is still missing some features.

There also exists an implementation for FreeBSD. This implementation has not yet been integrated into the kernel. It currently exists as a patch.

C.3 Library

C.3.1 Provided functionality

The library can send and receive DCCP packets in IP packets and can also use UDP packets to send and receive DCCP packets. The DCCP packets are valid and interact correctly with packets from other implementations. It can work on Operating systems which include support for DCCP at the same time as the kernel level implementation is in use. The library functions create a Unix socket that data can be read from and written to as with a normal socket, for a kernel level implementation of a network protocol. The functions provided by the library are based on the socket API and use the Unix socket that is used to send and receive data. In order to allow an unmodified program to use the library a version of the library is provided that can be preloaded before other libraries that implements the socket API using USDL for DCCP packets and OS socket API for non DCCP sockets.

User Program Interface

The interface functions take a file descriptor(FD), this FD is the FD for the Unix data socket used for reading and righting data to the socket, it is used as a key into a hash map to retrieve a data structure that holds the connection's control socket. The other arguments to the function are then sent to the library process, using the control socket, which calls a function that performs the action and then sends back the result.

There is also another library to be preloaded before other libraries, to allow unmodified programs to use the library rather than Linux kernel implementation. This library only includes the part of the library that runs in the user process and communicates with the other half of the main library.

OS

libpcap is used to receive DCCP/IP packets and raw sockets are used to send DCCP/IP packets. Packets sent using the Raw Sockets interface are valid DCCP packets inside of IP packets and compatible with the implementation of in the Linux kernel. A filter set on the incoming socket so that only DCCP packet for open ports are received by the library.

When a port is opened the existing filter is altered to include the new port and the filter is added to the incoming socket. When a packet is received, the connection to which it belongs is retrieved. The packet and connection are then passed to a function to update the state of the connection. After the connection has been updated another function is called to attempt to send a packet.

Each DCCP connection requires several timers. The OS part includes a list of functions to call after specific times, a function is included which can add a function to be run at any time. When a packet arrives or 10 ms passes the list of times is examined and the functions for the expired timers are run.

The library can also use UDP packets to carry DCCP packets this is done using the normal UDP packet sockets provided by the OS. Using UDP packet for transporting DCCP packets has the disadvantage that it increases the header overhead it does however have some advantages. It allows the library to be used in situations were it is not possible to use the Raw Sockets interface to send out DCCP packet and/or it is not possible to get incoming DCCP packets. Network address translation is used on a number of networks on the Internet, this involves, at least the sources or destination IP address of the packet to be changed, as the DCCP header checksum includes a pseudo header similar to TCP and UDP, this will make the packet invalid. As a result of this NATs need to know about the transport level protocols. Currently the Linux kernel has support doing NAT with DCCP however this is not widely used. Using UDP packets to transport DCCP packets would allow them to pass through a NAT that was setup to deal with UDP packets.

The library can be used on systems that include support for DCCP and also on operating systems that do not include support for DCCP. On Linux when the kernel supports DCCP, the firewall rules are changed to prevent the kernel from responding to packets intended for the library. The operating system and library share the port range, in

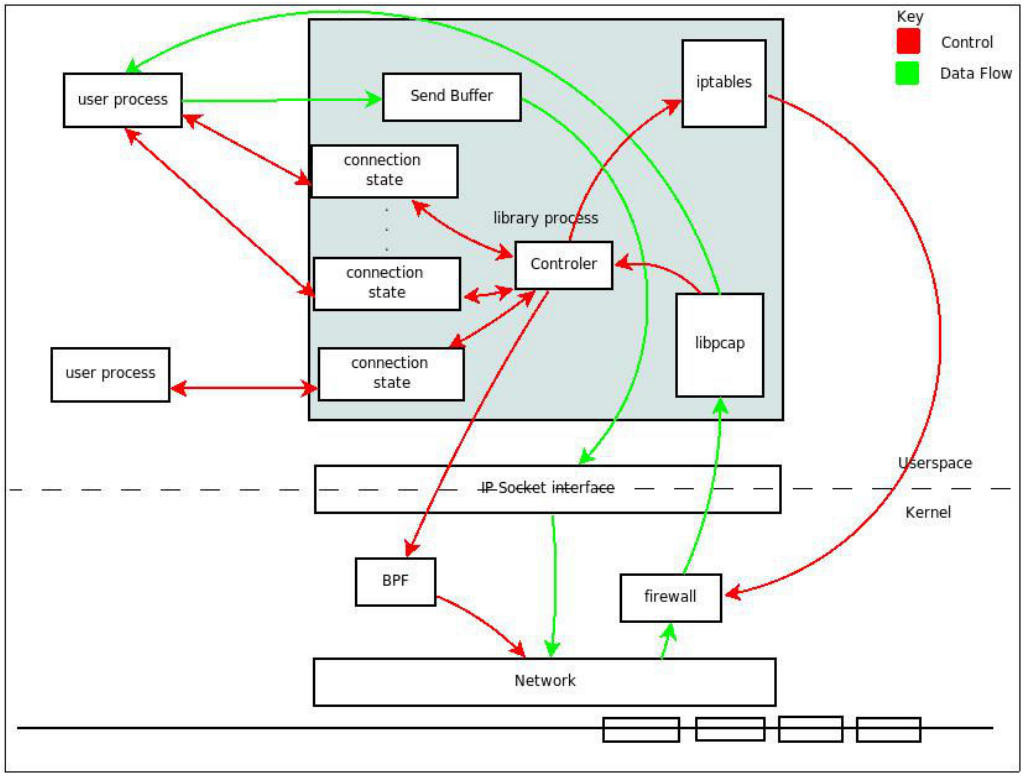


Figure C.1: The overall structure of DCCP

order to prevent the both using the same port the library attempts to create a DCCP socket for every socket when a socket is bound to a port and then attempts to bind the created socket to the port.

C.3.2 Structure

The library forms two parts, one of which is part of the user program and the other which runs in a separate process (figure C.1). Communication between the user programs processes and the library process is done through Unix sockets, the control of the DCCP sockets is done through byte stream sockets and data is moved through ordered packet oriented sockets, this allows the library to work when the user program creates more processes as the sockets will then be duplicated and sockets will only appear closed when there are no more references to them.

When a program tries to use the library and it is not already in use, the program will attempt to connect to the process and fail. It will then create the library process. The process will then connect to this new process.

For each connection a data structure is held that stores the state of the connection and the queues. The state of each feature is stored, options to be sent out on the next non-data packet are queued. Figure C.2 shows the relationship of this data structure to other. A hash map of port numbers to connections is stored containing opened or opening client ports and the listening server ports. The connections received on each server port are stored in a hash map for the remote port and address in the listening socket.

As the library cannot directly access the network hardware packets must be sent to the operating system to be sent out. When a user program sends data on the UNIX socket to be sent to the other end of the DCCP connection it is first read off of the UNIX socket and queued before an attempt is made to send it. Copying data in memory from one buffer to another is time consuming so when data put into the queue sufficient space is reserved to construct the packet headers, this decreases the number of memory copies. The data packet is then sent on to the operating system. Implementing the network stack and using UNIX socket for communication results each piece of data that is to be sent being written to a socket twice, once by the user program and once by the library process, for the reason the implementation is designed to minimise the number of other copying that is done. When data is received it is read out of the packet and sent directly to the user program using the UNIX socket.

The delay between packets being received and them being processed is important to the performance of a network protocol. The library being in user-space introduces extra delay into this process. To allow for the delaying the

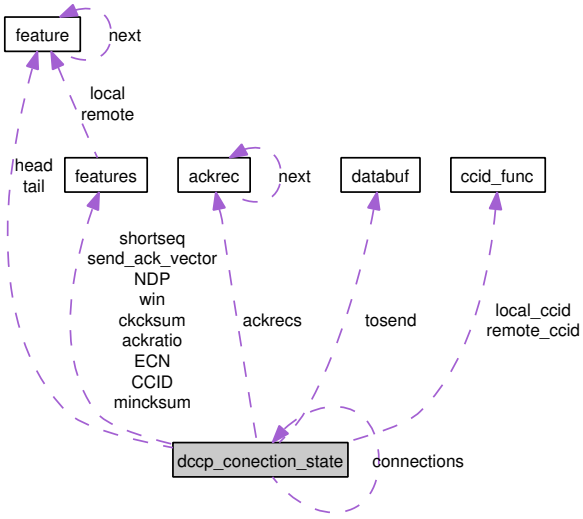


Figure C.2: The structure of the DCCP connection state struct

transmission of packets DCCP allows the time elapsed between a packet and its acknowledgement to be sent, this should reduce the impact of this extra delay. The delayed sending of packets requires timers that go off at the correct time to send packets at the correct time, the accuracy of these timers is also affected by the being in user-space.

A single process handles all of the connections. It consists of a loop that reads from a number of sockets and deals with values read. The incoming packets are read from PCAP packet capture devices. Commands for the sockets are read from the control sockets and the `dispatchComand` function is called to deal with the commands. The data is read from the data sockets. This can result in a large number of FDs to read from, to manage this, poll is used. select was initially used for waiting on the sockets, however it is limited to 1024 sockets, which would limit the library to around 500 connections.

The library has been designed so that it can work with multiple process, both one program that forks another process and several separate programs. This is accomplished by having the connection handling code in the library process, as a result when a process forks the operating system duplicates the file descriptors that the process had. The library keeps a small amount of information about the socket in the processes address space, this is not a problem as the memory of a process is also duplicated when a process forks. When one of the processes closes the socket that processes copy of the FD will be closed and the data structure freed, this will not affect the FDs of other processes or the data structure. The library will only detect that the socket has been closed when the last instance of the other end has been closed, the other closes will as a result not cause any reaction by the library process. This is consistent with the expected behaviour of sockets where only the last close would result in the connection being closed. It would be possible to use a more complicated solution were each process maintained separate connections to the library process for each DCCP connection, this would reduce the possibility of any problems caused by several processes using the same UNIX socket, however it would be considerably more complex to implement and would result in a larger number of sockets being required. The library could also have been implemented using a single process with a process handling the DCCP connection, this have the advantage that both process would share a single address space so reducing the number of memory copies that need to be performed. Using a thread would have the disadvantage of only allowing the program to have a single process.

The congestion control mechanisms are designed so that there is a structure which points to all the functions of the congestion control system. This limits the amount of code that is specific to anyone congestion control system, to the system itself and the code that sets the congestion control system. The congestion control system could alternatively have been more integrated into the rest of the library, this would have allowed unnecessary calls to function that don't do anything to be eliminated potentially allowing the implementation to be more efficient. This alternative would however make it far more complicated and time consuming to create new congestion control systems for the library.

When data is received from the user program it is first queued, before the congestion control mechanism is queried to determine if the packet can be transmitted. The attempt send function performs the process of examining the queue and calling the `requestSend` function of the congestion control mechanism, if it is successful in sending a packet it will repeat the process until there is no more queued data or the congestion control system does not allow any more

data to be sent. When the congestion control mechanism allows it, this allows more than one packet to be sent out at a time. It also gives preference to the connections that are limited by the available bandwidth rather than the speed at which the program is attempting to send data. The alternatives to using a queue in this manner would be to: 1) Send one or more packets at a time but read them directly from the data socket. This would have the advantage of not using pieces of memory for the queue. This would however allow one connection to consume too much of the time. A counter could also be included to limit the number of packets a connection can send in one attempt, but this would still fever the connections that produce the largest amount of data, 2) Send only one packet at a time. This would be the simplest solution requiring no memory management however this would also give more time to connections that are less limited by the congestion control system.

C.3.3 Congestion control system

There are currently two congestion control systems implemented. Both follow the same design were there is a setup function that is called when the congestion control system is to be initialised this function puts pointers to its functions into the passed connection structure and initialises memory for it's self to use and puts a pointer to this in the connection structure as well.

TCP-like Congestion Control

CCID 2 [57] is designed to approximate the behaviour of the congestion control system used by TCP [160, 4]. A congestion window is maintained to control the number of packets that are in the network at anyone time, as DCCP is a packet oriented rather than byte oriented, like TCP, the congestion window is in units of packets rather than bytes. TCP does not explicitly provide congestion control of the acknowledgements, though a change in the rate of data packets will also change the number of ACKs. CCID 2 provides for the ratio of ACKs to data packets to be changed when the loss of ACKs is detected. CCID 2 requires that acknowledgements contain an `ackvector` option that indicates the arrival status of packets.

This provides congestion control that rapidly changes rate at which packets can be sent in order to take advantage of, and react to, changes in the available bandwidth. This is useful to applications that can react quickly and to applications that do not require data to be transmitted at a continuous rate, but instead need a large amount of data to be transmitted.

TCP-Friendly Rate Control

CCID 3 [58] is a slight variation of TFRC [72] for DCCP. It is designed to compete fairly with TCP whilst providing a more smooth packet sending rate. It uses the following formula to determine the correct sending rate:

$$X = \frac{s}{R * \sqrt{2 * b * p/3} + (t_RTO * (3 * \sqrt{3 * b * p/8} * p * (1 + 32 * p^2)))} \tag{C.1}$$

X is the rate at which data is to be transmitted at in bits/second.

s is the packet size in bytes.

R is the round trip time in seconds.

p is the loss event rate, between 0 and 1.0, of the number of loss events as a fraction of the number of packets transmitted.

t_RTO is the retransmission timeout value in seconds that TCP would have calculated.

b is the number of packets acknowledged by a single TCP acknowledgement.

This system is intended to be more useful to the large number of applications need to be able to transmit packets of a fixed size, at rate that change smoothly. Fortunately a large number of streaming applications can operate in this fashion.

Of the two congestion control systems, CCID 3 is the more complex of the two. The method of controlling congestion that it is based on has been used by other protocols as a result there are existing implementations of it. These libraries were not used in the implementation of the library, this allows the implementation to be specific to DCCP rather than attempting to make the library fit the slightly modified version used by DCCP. If such a library had been used it could potentially have allowed the congestion control system to be implemented more quickly, though this is not necessarily true. The library implementation would have needed to be integrated into the DCCP implementation the complexity of doing this would determine if using a library was would have been quicker.

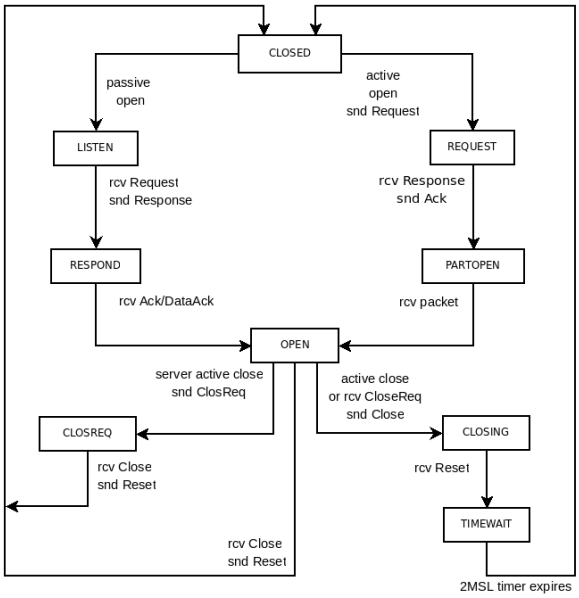


Figure C.3: The state transition diagram for DCCP

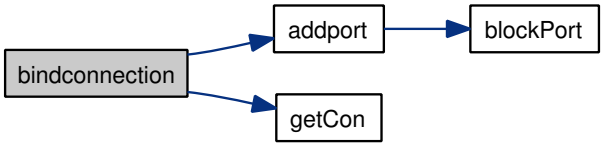


Figure C.4: The call graph of the bindconnection function

C.4 Process

C.4.1 Socket creation

The state transition diagram for DCCP is shown in figure C.3. When a connection is created it is in the CLOSED state. A socket that is to be used as a server socket must then be bound to a port and address, this will not change its state. The address structure is first sent to the library process, where it is passed to the `bindconnection` function, figure C.4 shows the call graph of this function. The `bindconnection` function first checks that the address structure is valid and returns an error if it is not. It then attempts to create a DCCP socket and if successful attempts to bind it to the address, this will prevent other process that are not using the library from binding connections to the same address and port. Once it is known that the kernel does not support DCCP or the port is not in use the connection structures local port and address are set to the supplied values, otherwise the function returns the appropriate error. The port is then added the map of ports to connection structures and the `addport` function is called. The `addport`function adds the port to the filter on the pcap socket and then adds a firewall rule to drop packets to that port, preventing the operating system reacting to packets on that port.

A server will then call listen on the socket. The listen function on the library process changes the state to listen and changes the socket's role to listen. A socket in the listen state is ready to receive incoming connection. When a listening socket receives a Request packet its header are first parsed, the new connection data structure is created from the packet and the parent connection structure, including any options set on the parent connection. This new connection structure is added the server socket's map of remote addresses to connections, the connection is then put into the `RESPOND` state and a Response packet is sent. A connection in the `RESPOND` state can receive `ACK` or `DATAACK` packets, when one of these is received the connection is put into the `OPEN` state.

A client socket will not need to be bound by the program though it can be if this is required. When the connect function is called the server's address is sent to the library process which calls `activeopen` function. The callgraph for `activeopen` is shown in figure C.5. The `activeopen` function begins by checking if the socket is already bound to

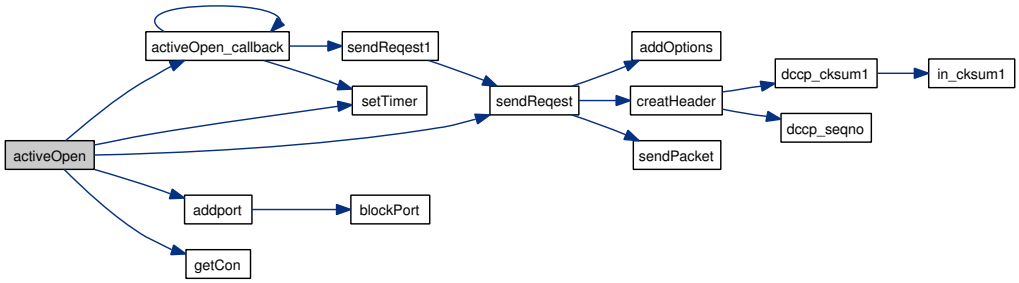


Figure C.5: The call graph of the activeopen function

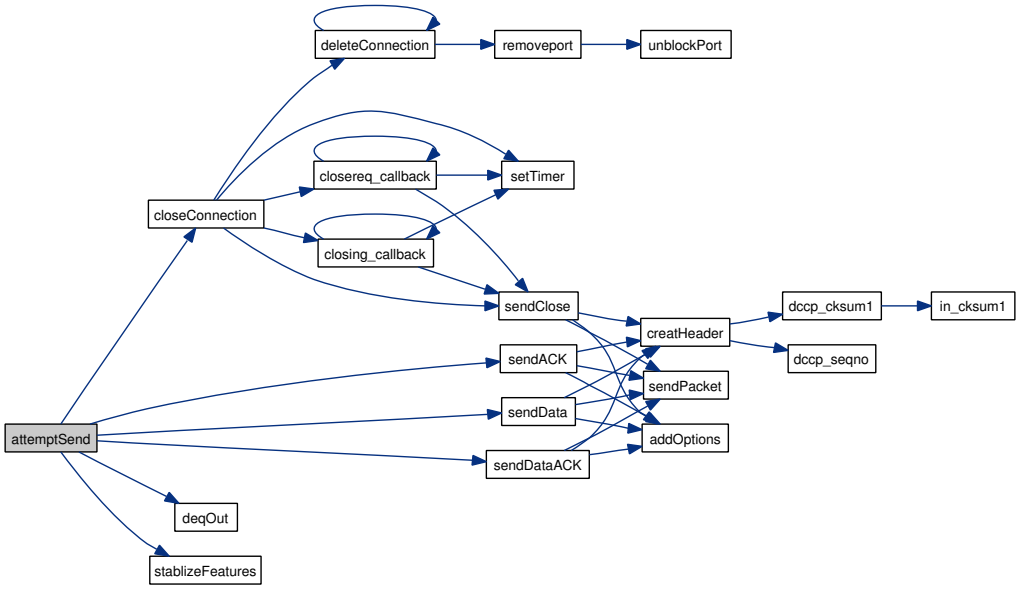


Figure C.6: The call graph of the attemptsend function

a port, if it is not then a random port is chosen, it is checked that it is larger than 1024 and that it is not in use by any other sockets, by searching the hash map and then creating and binding a DCCP socket, to prevent the operating system from using the port number, if any of these checks fail a new port is chosen and the process repeated. When a port has been selected the connection is added to the port map and addport is called. The remote address and port of the connection is then set to the supplied values, an initial sequence number is selected. A Request packet is then sent, an `activeopen` timer is started and the connection is put into the `REQUEST` state. When `activeopen` timer goes off it will send another request packet if the connection is still in the `REQUEST` state, it will then double the delay and set the timer again.

C.4.2 Sending data

To send a data packet using the library a user process simply writes the data to the socket that was returned to it by the socket function. A DCCP implementation must then send a `DATA` or `DATAACK` packet when this is permissible by the congestion control system. The library begins this process by reading the data from the data socket and putting it onto a queue, assuming there is space if there is not the data is left on the socket until there is. The attempt send function is then called, figure C.6 shows the call graph of that function. This function first checks that there is data on the queue to be sent. It then calls the congestion control system's `requestSend` function, this function differs between the different congestion control systems, but its purpose is to determine if it currently permissible to send a data packet. The `queueoptions` function of the receiving directions congestion control system is then called, this function queues up any options that the congestion control system intends to send, this function also indicate whether it requires the packet has acknowledgement number or not. A `DATA` or `DATAACK` packet will then be sent, after the packet is sent the process is repeated until there is no more queued data or the congestion control system will not allow more packet to be sent.

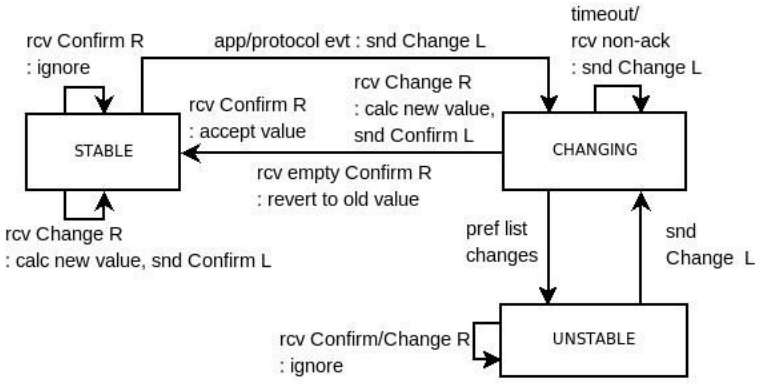


Figure C.7: The feature state transition diagram for DCCP

C.4.3 Recive data

When data is received by a DCCP connection the congestion control system is informed, the packet is marked acknowledgeable and then it is passed to the user process, in the case of this library the data is sent through the data socket. Section C.4.9 shows the code that is executed when a packet arrives.

C.4.4 Connection close

When a connection is to be closed the data and control sockets for it are closed, this causes the main loop function to detect this and begin the connection close. When the client wishes to close the connection it sends a `CLOSE` packet and moves into the `CLOSING` state, the server then receive the `CLOSE` and send back a `RESET` packet and move the connection into the `CLOSED` state. A server will send a `CLOSEREQ` packet and move into the `CLOSEREQ` state, this will result in the client sending a `CLOSE` packet and moving into the `CLOSING` state, the server will then send a `REST` packet and move into the `CLOSED` state. In both cases the client will move into the `TIMEWAIT` state after receiving a `RESET` packet. The library starts the process of closing the connection by removing the connection from the file descriptor list and then marking the end of the queue to indicate that the connection is to be closed. When the end of the queue is then reached the `attemptsend` function calls the `closeConnection` function. The `closeconnection` function determine if the connection is a client or server connection and then sends the appropriate packet and sets the corresponding timer.

C.4.5 Features

The protocol DCCP has a concept called features, which are options which negotiated between the two ends of the connection. Each feature starts with a default value, when either side changes its desired values it sends out a change l or a change r option, to indicate this, the other end then replies with a confirm which indicates it preferences and the value, if any, that has been agreed upon. Each congestion control mechanism can have its own features. The value and preference for both directions of a feature are stored in one structure. When a feature is to be changed the features data structure is located and then the desired value is copied into the structure and its state is changed to changing, if it is stable or unstable if it is already changing, a variable is then set to indicate that a feature change has been requested and it is added to queue of feature change options to be sent. When a packet that can carry feature change options is next to be sent the queue of feature changes is examined and all those feature changes that are still in the changing or unstable state are added to the packet. When a conformation option is received for a feature change that has been sent the feature is changed and the feature enters the stable state. When the other end of the connection sends a change option the new value is calculated and a confirm option is sent back, the state of the feature is then set to the stable state. If both sides wish to change a feature at the beginning of the connection then it will only be necessary for one end of the connection to send a change option. The feature change options could alternatively have been directly queued for the next packet. This would have had the advantage of being simpler and less time consuming. The advantage of using a queue and checking that the feature are still changing or unstable is that it avoids sending feature change requests when the negotiation has already happened. The state transitions for a feature are shown in figure C.7.

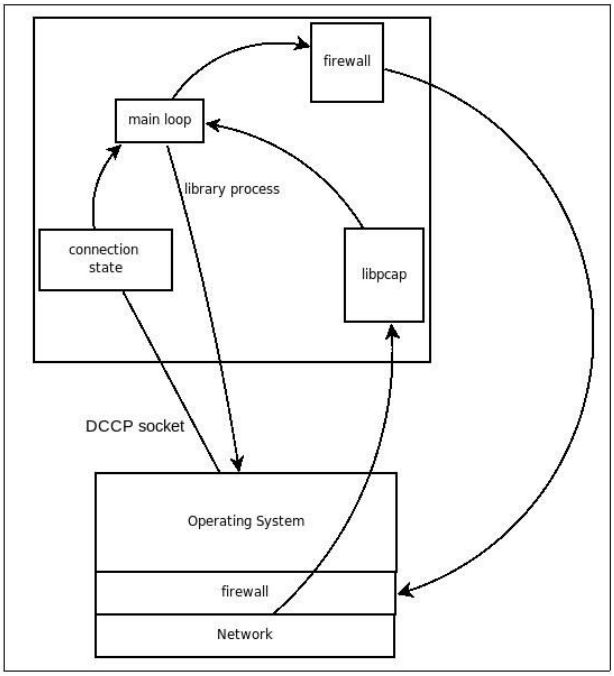


Figure C.8: The state transition diagram for DCCP

Feature change

When a feature is to be changed the features data structure is located and then the desired value is copied into the structure and its state is changed to changing, if it is stable or unstable if it is already changing, a variable is then set to indicate that a feature change has been requested and it is added to queue of feature change options to be sent. When a packet that can carry feature change options is next to be sent the queue of feature changes is examined and all those feature changes that are still in the changing or unstable state are added to the packet.

ParseFeatures

The parse features functions deals with incoming feature change and confirm options on packets. It follows the pseudo code given in the RFC, first the data structure representing the feature is located. The steps described are then performed as described for change and confirm R, the last two steps are then repeated for change and confirm L. The data structure is located by the `findFeature` function, which first uses a switch statement to check the general features then a function is called in one of the congestion control systems to locate a feature if one exists. The validity of feature change options is also checked by a function similar to `findFeature`, there are also functions calculate the new feature value and process the change in feature value. The feature data structure contains a variable that indicates which reconciliation rule is to be used for that feature, the `calculateValue` function reads this variable and performs the reconciliation, it then queues a confirm option and returns the new value.

C.4.6 Main loop

Algorithim 1 shows t he pseudo code for the main loop.

C.4.7 DispatchCommand

The `dispatchcommand` function deals with calling the correct function with the correct arguments when commands are received from the user process on a control socket.

In order to send back error numbers to the user process the number that `errno` would normally be set to is multiplied be negative one and set as the return value. For the functions were some of the parameters are used to return values, pointers to the command buffer are passed to the function. The function then simply writes the value to the memory pointed to and the command buffer is then sent back to the user process that then reads the value out of the buffer.

```
while true do
    Add the file descriptors to the array;
    Add the pcap interfaces;
    Add the listening UNIX socket;
    Add the control sockets;
    Add the data sockets;
    Poll on the sockets;
    if Poll returned -1 then
        print error and exit;
    else
        Check the PCAP interfaces and process any packets;
        if Listening socket can be read then
            Accept any incoming connections from a user process;
            Accept a new control socket;
            Add the socket to the list of control sockets;
            Create a new socket data structure;
            Create the socket pair for data;
            Add the new connection to the list;
            Send the FD of the data socket to the user process;
            Close the user process's end of the socket pair;
            Put the socket in non blocking mode;
            Add the connection to the control sockets data structure;
        forall Control sockets do
            if the connection has been closed then
                remove the connection;
            else if there is an incoming command then
                process the command;
        forall Data sockets do
            if The data socket can be read then
                if There is space on the queue then
                    if the other end of the data connection has been closed then
                        close this end of the data connection;
                        Remove the connection from the list of file descriptors;
                        Queue up a connection close to be sent;
                        Attempt to send data or a close;
                    else if there is data that has been read then
                        Queue up the data to be sent;
                        Attempt to send data;
                else if there is no space on the queue then
                    mark the queue as full;
```

Algorithm 1: The pseudo code for the main loop of the library process.

```
switch the first int of the command buffer do /* to Determine what type of command it is */
    case BINDSOCKET /* bind */
        if connection != NULL then
            extract the function arguments from the command buffer;
            call the function that performs bind;
        send back the return value;
    case SETLISTEN /* listen */
        if connection != NULL then
            extract the function arguments from the command buffer;
            call the function that performs listen;
        send back the return value;
    case CONNECT /* connect */
        if connection != NULL then
            extract the function arguments from the command buffer;
            call the function that performs connect;
    case SETSOCKOPT /* setsockopt */
        if connection != NULL then
            if the socket option is a CHANGE_L or CHANGE_R then
                Change the passed data structure so that its desired value pointer points to the correct place in
                the command buffer;
            extract the function arguments from the command buffer and pass them to a function that performs
            setsockopt;
        send back the return value;
    case GETSOCKOPT /* getsockopt */
        if connection != NULL then
            extract the function arguments from the command buffer;
            call the function that performs getsockopt;
        put the return value into the command buffer;
        Send back the now changed command buffer;
    case GETPEERNAME /* getpeername */
        if connection != NULL then
            extract the function arguments from the command buffer;
            call the function that performs getpeername;
        put the return value into the command buffer;
        Send back the now changed command buffer;
    case GETSOCKNAME /* getsockname */
        extract the function arguments from the command buffer and pass them to a function that performs
        getsockname;
        put the return value into the command buffer;
        Send back the now changed command buffer;
```

Algorithm 2: The dispatch command function.

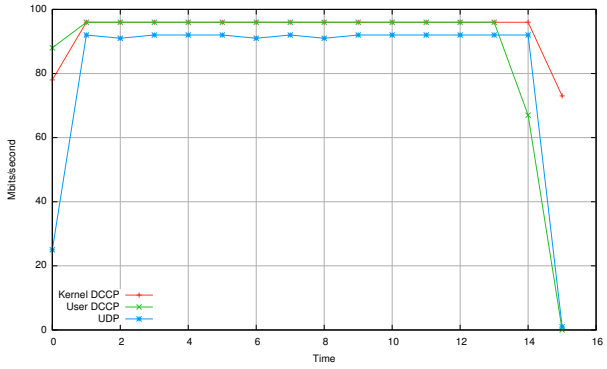


Figure C.9: The number of bytes per second for a library DCCP connection, sampled once a second.

The library is designed to work on systems were there is a kernel level implementation, to achieve this the kernel must be prevented from using the same port as the library and it must be prevented from receiving packets intended for the library. To prevent the operating system using the same ports as the library, the library attempt to create a DCCP socket when it is to bind a port and then binds it the port that is to be used. The library then adds a rule to the firewall to drop packets sent to the port that the library is to use. When the socket is closed the kernel DCCP socket is also closed and the firewall rule is removed. Alternatively the library could have been designed not to worked on a system where there was an existing implementation, this would have been simpler, though would have reduced the usefulness of the library. The library could also have been designed to work on a system with DCCP, but not at the same time as the system DCCP socket were being used, this would have been achieved by creating a firewall rule to drop all DCCP packet.

C.4.8 Decisions

When data is received from the user program it is first queued, before the congestion control mechanism is queried to determine if the packet can be transmitted. The attempt send function performs the process of examining the queue and calling the requestSend function of the congestion control mechanism, if it is successful in sending a packet it will repeat the process until there is no more queued data or the congestion control system does not allow any more data to be sent. When the congestion control mechanism allows it, this allows more than one packet to be sent out at a time. It also gives preference to the connections that are limited by the available bandwidth rather than the speed at which the program is attempting to send data. The alternatives to using a queue in this manner would be to: 1) Send one or more packets at a time but read them directly from the data socket. This would have the advantage of not using pieces of memory for the queue. This would however allow one connection to consume too much of the time. A counter could also be included to limit the number of packets a connection can send in one attempt, but this would still fever the connections that produce the largest amount of data, 2)Send only one packet at a time. This would be the simplest solution requiring no memory management however this would also give more time to connections that are less limited by the congestion control system.

A single process handles all of the connections. It consists of a loop that reads from a number of sockets and deals with values read. The incoming packets are read from PCAP packet capture devices. Commands for the sockets are read from the control sockets and the dispatchComand function is called to deal with the commands. The data is read from the data sockets. This can result in a large number of FDs to read from, to manage this, poll is used. select was initially used for waiting on the sockets, however it is limited to 1,024 sockets, which would limit the library to around 500 connections.

C.4.9 updateconnection

The `updateConnection` function update the state of the connection when a packet is received. This function follows the pseudo code given in RFC 4340, however there are places where it differs. The following is the differences from that pseudo code.
connection = con /* The connection that this packet belongs to */
connection1 = NULL /* The parent connection of the connection */

hdrlen = dccp->dccph_doff*4; /* The length of the DCCP headers and options not yet parsed */
seq = 0 /* The packet sequence number */
ackSeq = 0 /* This packet ack sequence number */
/* Finish parsing the standard headers and reduce the value of hdrlen accordingly */

if(connection state == LISTEN)
 /* Check if the packet belongs to an already accepted connection */
  connection = the already accepted connection from the connection's hash map
  connection1 = con

Find the data part of the packet

Find the packet header structures
if an ack header has been found
 ackSeq = the ack number

If the connection is in the timewait state and this packet is not a reset send a reset and return

Step 3 of the RFC but first a new connection state is created and the values of the parent copied into it, it is added to its parent connections hash map and connection is set to the new connection's state and connection1 is set to the old value of connection
Steps 4 – 8 are then the same as the RFC
Steps 9 is also the same except: if the connection is a client connection has not yet reached the `PARTOPEN` state the user process is sent an error message. In all cases the connection is removed from the list of file descriptors and the data connection is closed.
Steps 10 After the rest of step is completed the user process is informed that the connection has now reached the `PARTOPEN` state
Steps 11 – 16 are then the same as the RFC

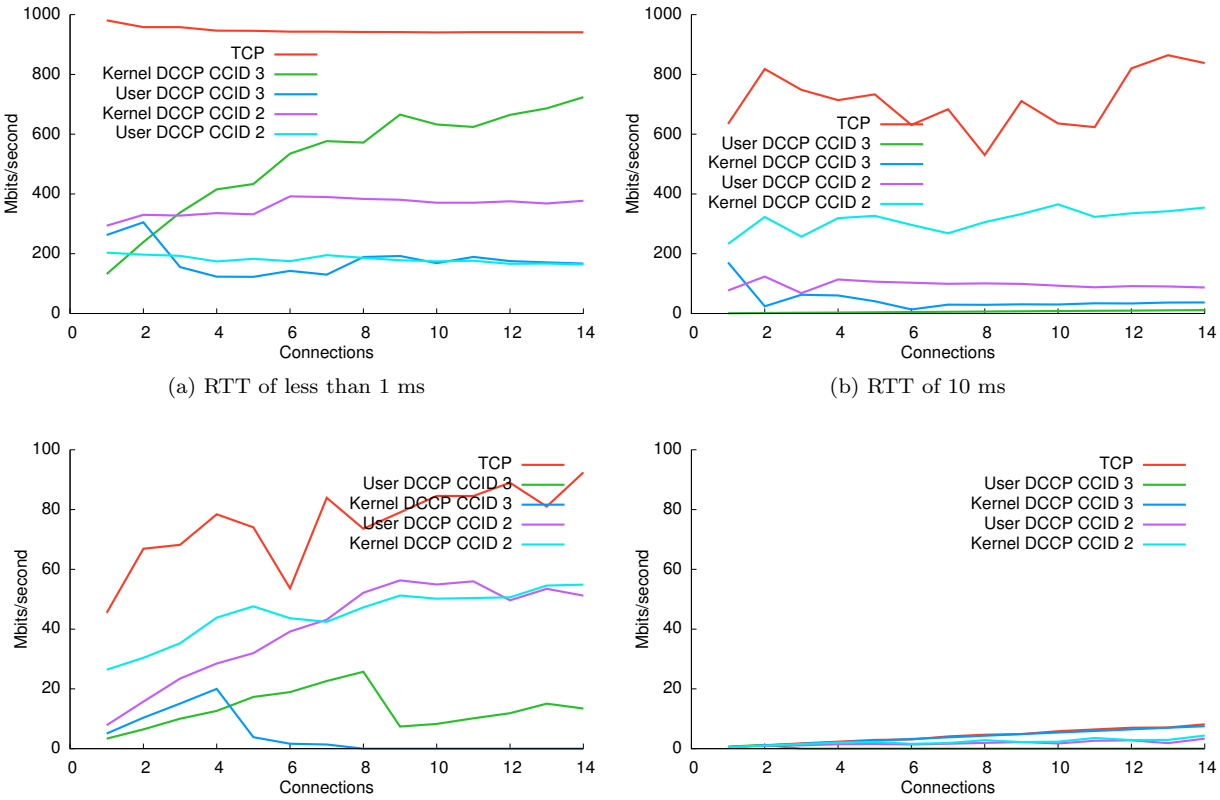
Unix sockets are used for sending and receiving data between the user process and the library process, this allows the usual functions for working with sockets work. It also has the consequence that if the user program crashes or is killed the library will correctly close down the connections – the original design of the library used a thread, if program crashed the connection was left in an open state and did not respond to further packets.

C.4.10 Connection

The largest part of the library is the connection part which contains the functions for updating the connection state. As each packet arrives the update connection function is called. This function processes the packet's standard headers and then passes the packet to a function, in the options part, that parses the options. `packet_callback` then calls the `attemptsend` function, in connection, this function checks to if there is any data packets to be sent, if there are the `requestSend` function of the congestion control system is called, if the returns true then the congestion control system for the other direction is given a chance to queue options and send a `DATAACK` packet instead of a `DATA` packet. If data cannot be sent then the receiving congestion control system gets a chance to send an `ACK`.

C.5 Evaluation

The library has been tested to ensure that it correctly implements DCCP as defined in the RFC and that it interacts correctly with Linux kernel implementation of DCCP. The performance of the library is also important as its congestion control systems would not be of any use if it were not able to send packets fast enough that it should be limited by the congestion control system. The performance of the library on its own has been tested to determine the limits of its performance when several connections were opened simultaneously. The congestion control system has been tested to work correctly when competing with other congestion controlled connections, both against kernel DCCP connections and TCP connections. The library should be able to operate correctly when transporting different types of traffic



(a) RTT of less than 1 ms

(b) RTT of 10 ms

(c) RTT of 100 ms

(d) RTT of 1 s

Figure C.10: The bandwidth utilisation against number of connections for the two DCCP implementations and both congestion control systems and TCP

including application where there is less data to send than can be sent. To test that operate correctly audio and video were streamed over user DCCP connections.

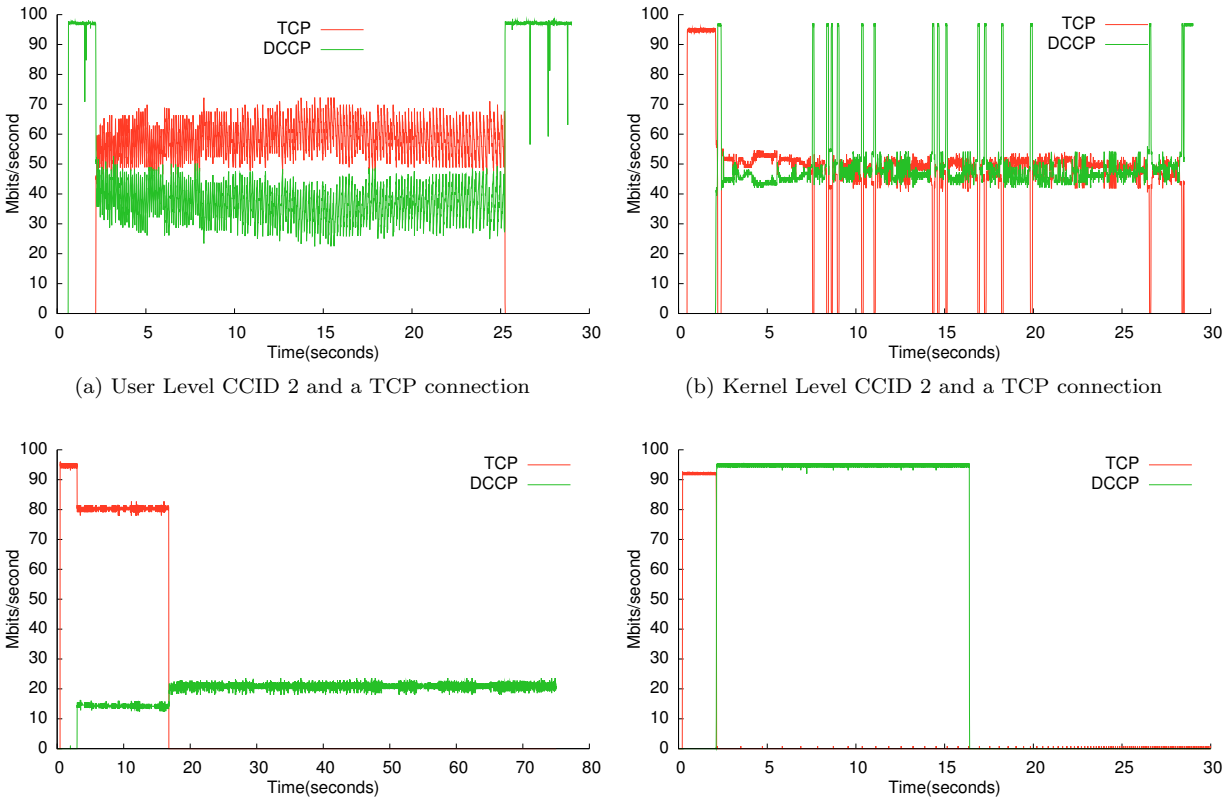
A network was created with three computers connected to a 100 Mbit switch. With two computers sending packets and one receiving them. A number of programs were created to test USDL.

The library has been tested using a number of programs. Connections were established and data was sent, the speed of the data transmission and the loss rate were compared to the Linux kernel implementation, packet trace were also examined to ensure that packet level interaction of a single connection was correct and that the congestion control systems did limit the transmission of packets. A program was created to accept a large number of incoming connection from another program, these were tested using the library and the Linux kernel implementation of DCCP to detect problems with large numbers of connections. The feature negotiation of the library was tested by changing the congestion control preferences in programs to ensure that the negotiation was performed correctly, in doing this it was discovered that Linux kernel implementation did not correctly implement this.

The bandwidth utilisation of three separate flows of traffic, a user DCCP connection, a kernel DCCP connection and a UDP flow are shown in figure C.9. The two DCCP connections similarly, with most of the difference being the result of sampling granularity rather than any difference in the transfer rate achieved. The UDP flow the amount of traffic arriving at the other end is less despite the fact that there is no congestion control system.

The bandwidth utilisation against number of connections for the two DCCP implementations and both congestion control systems and TCP is shown in figure C.10, this was recorded on a 1 Gbit connection. They show that user level CCID 2 is able to operate at around 200 Mbps. The top right hand graph shows user level CCID 2 is able to able use as much bandwidth as the kernel level CCID 2 for up to 14 concurrent connections. These graphs show that TCP performs better than DCCP at normally encountered round trip times, however when the round trip time increase sufficiently the performance of TCP decreases. The performance of CCID 3 when only competing against itself is better than that of CCID 2, when it competes against its self.

Both implementations competing against a TCP connection are shown in figure C.11. The top two graphs show that connections controlled by DCCP's TCP like congestion control are able to fairly share bandwidth with TCP



(a) User Level CCID 2 and a TCP connection

(b) Kernel Level CCID 2 and a TCP connection

(c) User Level CCID 3 and a TCP connection

(d) Kernel Level CCID 3 and a TCP connection

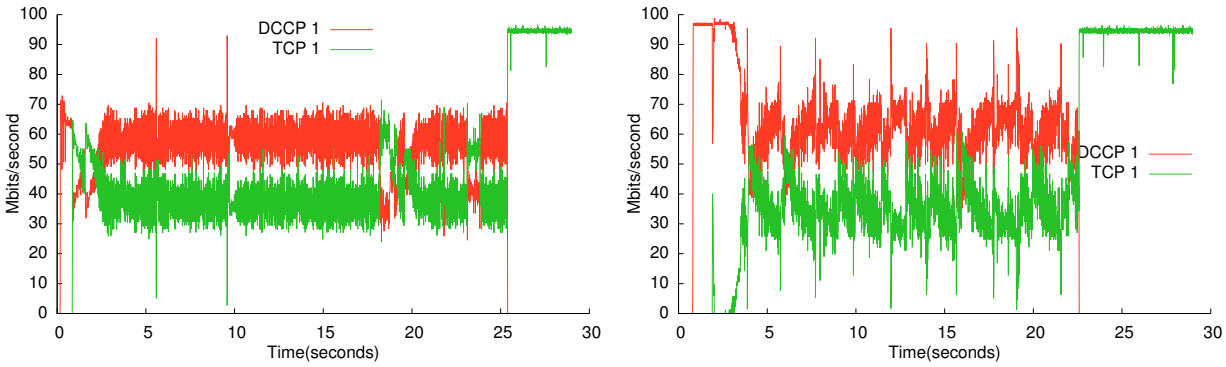
Figure C.11: The number of bytes per second for one DCCP connections and one TCP connection, sampled once every 10 milliseconds, on a network with no routers.

connections. The top left hand graphs shows a user level DCCP connection and the top right shows a kernel level DCCP connection. The user DCCP connections will not use more than there fair share of the bandwidth, however two connections never exactly share the same amount of bandwidth. User DCCP when competing against TCP in this setup always used less bandwidth than TCP when the two compete for bandwidth. The bottom two graphs show that connections controlled by DCCP's TCP friendly rate control have difficulty obtaining there fair share of bandwidth when competing with TCP connections, this tendency is more pronounced in the right hand graph which shows the kernel level implementation.

Graphs of DCCP and TCP connections on a network containing one router are shown in figure C.12. On this network DCCP CCID2 and TCP are also able to fairly share bandwidth. The top left graph shows User DCCP and the top right show the kernel DCCP. The user DCCP share bandwidth similarly well to when there was no router though it is the DCCP rather than the TCP that gets slightly greater than its fair share. The kernel implementation on the other hand, whilst still performing acceptably takes a larger amount of the bandwidth, and is less stable in its bandwidth utilisation.

DCCP and TCP react differently to changes in the round trip time, as a result the fairness between a TCP and a DCCP connection on the same link will be different depending on the RTT of that link. The fairness [86] between a TCP connection and a DCCP connection against the RTT of the link is shown in figure C.13a. In all of the cases the TCP connection takes more of the bandwidth than the DCCP connection. At a very short RTT the kernel implementation is able to achieve greater fairness than user space implementation, however when the RTT is greater than 1 ms the user space implementation achieves better fairness than the kernel implementation. The method of measuring fairness gives a value between 1/n and 1 were n is the number of connections, in this case 2. 1 would indicate a completely fair sharing of the available bandwidth and 0.5 would indicate that one of the connections is taking up all of the available bandwidth.

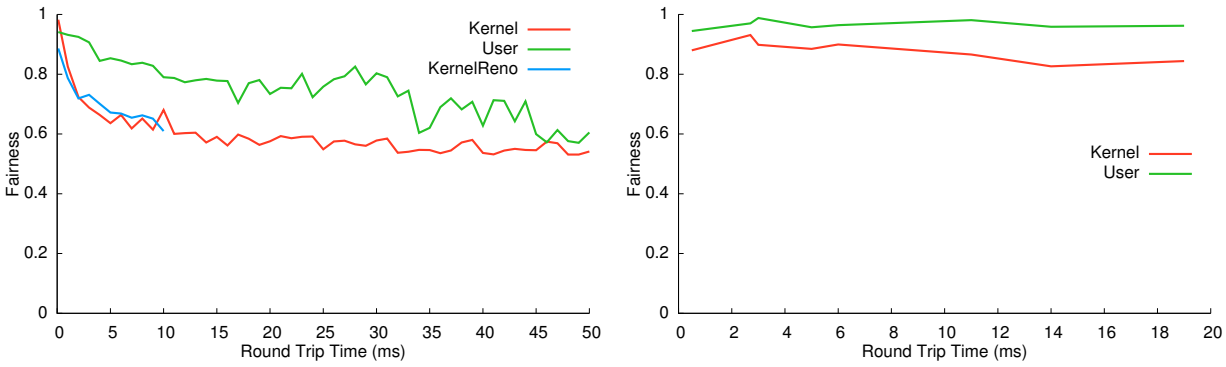
The mean fairness between DCCP CCID 2 connection and a TCP connection for both user and the kernel DCCP implementations, against the RTT are shown in figure C.13b. The router introduces greater delay into the network so it is impossible to get fairness values for a RTT less than 1 ms. For RTT greater than 1 ms connections achieve



(a) User Level CCID 2 and a TCP connection

(b) Kernel Level CCID 2 and a TCP connection

Figure C.12: The number of bytes per second for one DCCP connections and one TCP connection, sampled once every 10 milliseconds, on a network with 1 router.



(a) Without a router

(b) With a router

Figure C.13: The fairness between a TCP connection and a DCCP connection against the RTT of the link.

better fairness with a router than without.

Two kernel DCCP CCID 2 connections going through the same router and competing for bandwidth are shown in figure C.14a.

A kernel DCCP CCID 2 connection and user DCCP CCID 2 connection is shown in figure C.14b. This shows that the two implementations of DCCP are able to share bandwidth fairly between them.

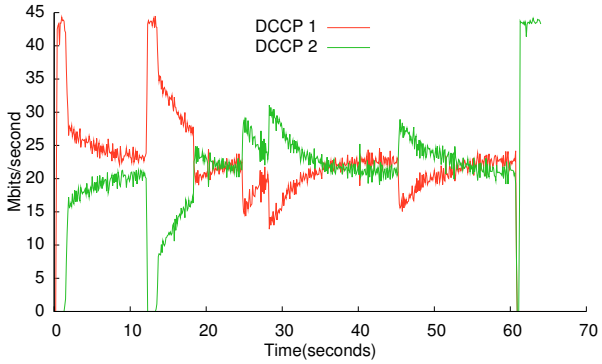
An audio file was transcoded into the Speex[1] format and streamed across a network. The bandwidth utilisation of the audio stream on a network with no other traffic, which results in predictable packet pattern, is shown in figure C.15a. The DCCP audio stream plus a shorter TCP connection are shown in figure C.15b shows, as expected the affect on the DCCP connection is very small as it is taking up less than its fair share of the shared bandwidth.

A kernel DCCP connection carrying RTP packets, that contain audio data, are shown in figure C.15a. This traffic is being sent over a connection with no other traffic.

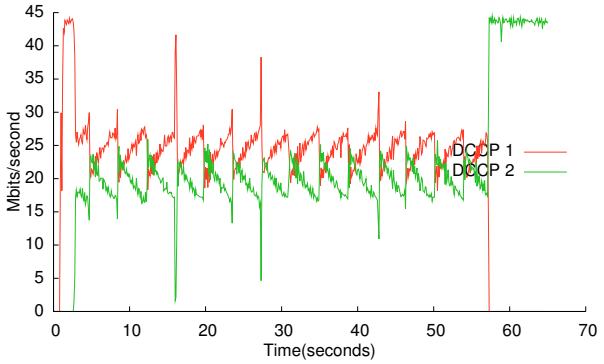
C.6 Conclusion

The library provides an API that is matches the Unix socket API and provides FD that can be used with the standard IO functions. This implementation of DCCP is valid and performs well at normally available bandwidths. It will allow new congestion control systems to be tested and new userspace interfaces developed.

[1]Xiph.Org Foundation and Jean-Marc Valin. Speex. `http://www.speex.org/`, 13 February 2002. [Online accessed 29-September-2010]

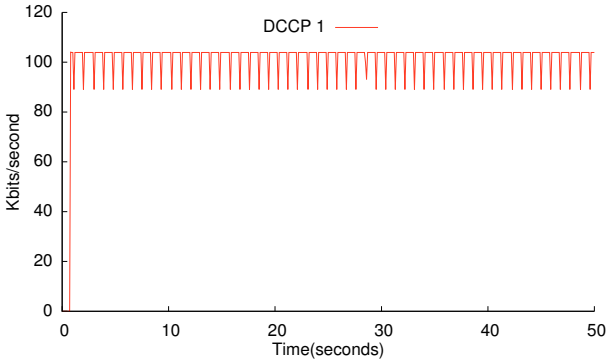


(a) Two kernel DCCP connections

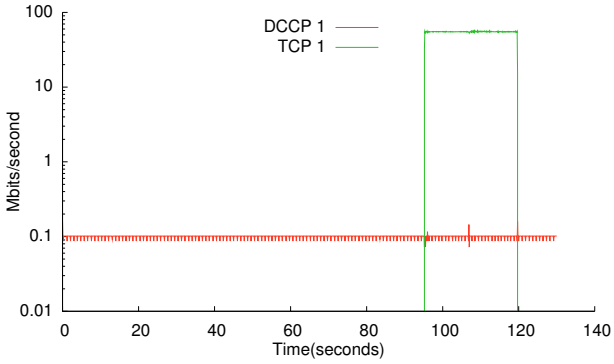


(b) One kernel DCCP connection and one USDL DCCP connection

Figure C.14: The bandwidth utilisation of two DCCP connections travelling through the same router.



(a) On its own.



(b) With a TCP connection.

Figure C.15: The bandwidth utilisation of a kernel level DCCP connection carrying an RTP stream of audio.

Appendix D

Tables

```
traceroute to 8.2.32.1 (8.2.32.1), 30 hops max, 40 byte packets
 1  router-cole-0.cs.st-andrews.ac.uk (138.251.194.1)  0.629 ms  1.161 ms  1.397 ms
 2  nhx-6513-jcb.st-and.ac.uk (138.251.2.193)  0.561 ms  0.546 ms  0.538 ms
 3  uos-gw.st-and.ac.uk (138.251.2.1)  0.523 ms  0.514 ms  0.505 ms
 4  194.81.114.45 (194.81.114.45)  1.313 ms  1.302 ms  1.554 ms
 5  so-2-3-2.leed-sbr1.ja.net (146.97.42.49)  8.032 ms  8.022 ms  8.011 ms
 6  so-2-2-0.lond-sbr4.ja.net (146.97.33.30)  12.705 ms  12.807 ms  12.796 ms
 7  ldn-b4-link.telia.net (213.248.100.237)  12.788 ms  12.773 ms  12.763 ms
 8  ge-6-22.car2.london1.level3.net (4.68.111.181)  12.755 ms  12.745 ms  13.138 ms
 9  ae-31-53.ebr1.London1.Level3.net (4.68.116.94)  20.312 ms  20.297 ms  20.286 ms
10  ae-1-100.ebr2.London1.Level3.net (4.69.132.118)  26.408 ms  22.892 ms  22.879 ms
11  ae-44-44.ebr1.newyork1.level3.net (4.69.137.78)  83.570 ms
12  ae-81-81.csw3.NewYork1.Level3.net (4.69.134.74)  88.541 ms  88.527 ms
13  ae-63-63.ebr3.NewYork1.Level3.net (4.69.134.97)  89.021 ms
14  ae-6.ebr2.dallas1.level3.net (4.69.137.121)  129.026 ms  120.080 ms  120.063 ms
15  ae-62-62.csw1.dallas1.level3.net (4.69.136.138)  123.886 ms
16  ae-1-69.edge3.Dallas1.Level3.net (4.68.19.8)  120.505 ms
17  gw-8-2-32-0-22.lindenlab.com (8.2.32.1)  122.706 ms  123.454 ms  122.682 ms
```

```
traceroute to 8.2.32.1 (8.2.32.1), 30 hops max, 40 byte packets
 1  router-cole-0.cs.st-andrews.ac.uk (138.251.194.1)  0.629 ms  1.161 ms  1.397 ms
 2  nhx-6513-jcb.st-and.ac.uk (138.251.2.193)  0.561 ms  0.546 ms  0.538 ms
 3  uos-gw.st-and.ac.uk (138.251.2.1)  0.523 ms  0.514 ms  0.505 ms
 4  194.81.114.45 (194.81.114.45)  1.313 ms  1.302 ms  1.554 ms
 5  so-2-3-2.leed-sbr1.ja.net (146.97.42.49)  8.032 ms  8.022 ms  8.011 ms
 6  so-2-2-0.lond-sbr4.ja.net (146.97.33.30)  12.705 ms  12.807 ms  12.796 ms
 7  ldn-b4-link.telia.net (213.248.100.237)  12.788 ms  12.773 ms  12.763 ms
 8  ge-6-22.car2.london1.level3.net (4.68.111.181)  12.755 ms  12.745 ms  13.138 ms
 9  ae-31-53.ebr1.London1.Level3.net (4.68.116.94)  20.312 ms  20.297 ms  20.286 ms
10  ae-1-100.ebr2.London1.Level3.net (4.69.132.118)  26.408 ms  22.892 ms  22.879 ms
11  ae-44-44.ebr1.newyork1.level3.net (4.69.137.78)  83.570 ms ae-43-43.ebr1.newyork1.level3.net (4.69.137.74)  85.107 ms ae-44-44.ebr1.newyork1.level3.net (4.69.137.78)  84.292 ms
12  ae-81-81.csw3.NewYork1.Level3.net (4.69.134.74)  88.541 ms  88.527 ms ae-71-71.csw2.newyork1.level3.net (4.69.134.70)  89.459 ms
13  ae-63-63.ebr3.NewYork1.Level3.net (4.69.134.97)  89.021 ms ae-93-93.ebr3.newyork1.level3.net (4.69.134.109)  87.842 ms ae-73-73.ebr3.newyork1.level3.net (4.69.134.101)  88.997 ms
14  ae-6.ebr2.dallas1.level3.net (4.69.137.121)  129.026 ms  120.080 ms  120.063 ms
15  ae-62-62.csw1.dallas1.level3.net (4.69.136.138)  123.886 ms ae-92-92.csw4.dallas1.level3.net (4.69.136.150)  124.004 ms ae-72-72.csw2.Dallas1.Level3.net (4.69.136.142)  122.507 ms
16  ae-1-69.edge3.Dallas1.Level3.net (4.68.19.8)  120.505 ms ae-3-89.edge3.Dallas1.Level3.net (4.68.19.136)  123.491 ms ae-1-69.edge3.dallas1.level3.net (4.68.19.8)  121.745 ms
17  gw-8-2-32-0-22.lindenlab.com (8.2.32.1)  122.706 ms  123.454 ms  122.682 ms
```

```
traceroute to 216.82.16.64 (216.82.16.64), 30 hops max, 40 byte packets
 1  router-cole-0.cs.st-andrews.ac.uk (138.251.194.1)  0.608 ms  1.022 ms  1.257 ms
 2  nhx-6513-jcb.st-and.ac.uk (138.251.2.193)  0.542 ms  0.530 ms  0.520 ms
 3  uos-gw-nhx.st-and.ac.uk (138.251.2.1)  0.507 ms  0.498 ms  0.488 ms
 4  194.81.114.45 (194.81.114.45)  1.489 ms  1.476 ms  1.467 ms
 5  so-2-3-2.leed-sbr1.ja.net (146.97.42.49)  8.284 ms  8.270 ms  8.260 ms
 6  so-2-2-0.lond-sbr4.ja.net (146.97.33.30)  12.624 ms  12.711 ms  12.696 ms
 7  ldn-b4-link.telia.net (213.248.96.37)  114.458 ms  114.448 ms  114.436 ms
 8  ldn-bb2-link.telia.net (80.91.250.238)  13.462 ms ldn-bb1-link.telia.net (80.91.249.77)  13.451 ms ldn-bb1-link.telia.net (80.91.250.234)  13.430 ms
 9  nyk-bb1-link.telia.net (80.91.249.249)  85.258 ms nyk-bb2-pos0-2-0.telia.net (213.248.65.94)  84.888 ms nyk-bb1-pos0-2-0.telia.net (213.248.65.90)  87.531 ms
10  nyk-b5-link.telia.net (80.91.248.146)  85.224 ms  85.077 ms  85.065 ms
11  4.68.63.213 (4.68.63.213)  83.681 ms level3-ic-120265-nyk-b5.c.telia.net (213.248.90.126)  83.798 ms 4.68.63.213 (4.68.63.213)  83.670 ms
12  vlan89.csw3.NewYork1.Level3.net (4.68.16.190)  87.095 ms vlan69.csw1.NewYork1.Level3.net (4.68.16.62)  85.643 ms vlan99.csw4.NewYork1.Level3.net (4.68.16.254)  96.827 ms
13  ae-93-93.ebr3.newyork1.level3.net (4.69.134.109)  85.685 ms ae-83-83.ebr3.NewYork1.Level3.net (4.69.134.105)  84.133 ms ae-73-73.ebr3.newyork1.level3.net (4.69.134.101)  85.352 ms
14  ae-6.ebr2.dallas1.level3.net (4.69.137.121)  121.139 ms  133.577 ms  131.446 ms
15  ae-62-62.csw1.dallas1.level3.net (4.69.136.138)  119.458 ms ae-82-82.csw3.dallas1.level3.net (4.69.136.146)  129.333 ms  126.402 ms
16  ae-22-79.car2.Dallas1.Level3.net (4.68.19.68)  161.756 ms ae-32-89.car2.Dallas1.Level3.net (4.68.19.132)  160.573 ms ae-42-99.car2.Dallas1.Level3.net (4.68.19.196)  160.887 ms
17  DATABANK-HO.car2.Dallas1.Level3.net (4.71.170.2)  121.998 ms  121.467 ms  122.161 ms
18  ae_cw_10g.databank.com (63.164.96.62)  121.366 ms aw_cw_10g.databank.com (63.164.96.54)  123.187 ms  122.623 ms
```

```
traceroute to 63.210.159.179 (63.210.159.179), 30 hops max, 40 byte packets
```

Packet Type	Frequency	Throttle	Number	Bytes	Mean Size	Max Size	Min Size	% of Throttle	% of total
PacketAck	Fixed	NoThrottle	2,802,003	134,775,425	48.10	1,043	43	90.26%	15.33%
PacketAck	Fixed	NoThrottle	148,062	6,566,458	44.35	1,043	43	4.40%	0.75%
StartPingCheck	High	NoThrottle	97,887	4,235,502	43.27	901	40	2.84%	0.48%
Remainder	N/A	NoThrottle	92,585	3,734,200	40.33	169	40	2.50%	0.42%
TransferPacket	High	Asset	89,682	92,673,227	1,033.35	1,226	66	92.09%	10.54%
InventoryDescendents	Low	Asset	6,524	5,806,484	890.02	1,361	127	5.77%	0.66%
ConfirmXferReply	High	Asset	16,800	791,582	47.12	68	47	0.79%	0.09%
Remainder	N/A	Asset	9,889	1,357,218	137.25	1,293	47	1.35%	0.15%
ImprovedTerseObjectUpdate	High	Task	2,583,600	431,054,400	166.84	1,393	93	26.19%	49.02%
AgentUpdate	High	Task	2,629,554	366,260,349	139.29	162	111	22.25%	41.65%
ViewerEffect	Medium	Task	821,950	294,429,213	358.21	1,086	94	17.89%	33.48%
ObjectUpdateCompressed	High	Task	127,204	128,644,023	1,011.32	1,496	164	7.82%	14.63%
ObjectUpdate	High	Task	283,071	121,788,074	430.24	1,389	141	7.40%	13.85%
CoarseLocationUpdate	Medium	Task	280,371	79,756,182	284.47	1,486	42	4.85%	9.07%
AvatarAnimation	High	Task	581,009	56,251,387	96.82	235	91	3.42%	6.40%
SimStats	Low	Task	132,756	37,569,948	283.00	283	283	2.28%	4.27%
ViewerEffect	Medium	Task	188,107	29,888,924	158.89	234	124	1.82%	3.40%
SendXferPacket	High	Task	16,808	17,590,913	1,046.58	1,226	51	1.07%	2.00%
AttachedSound	Medium	Task	194,494	17,310,036	89.00	102	89	1.05%	1.97%
ObjectUpdateCached	High	Task	54,477	14,242,182	261.43	1,042	58	0.87%	1.62%
Remainder	N/A	Task	512,988	51,181,208	99.77	1,348	36	3.11%	5.82%
ImageData	High	Texture	73,073	47,264,553	646.81	660	200	1.80%	5.38%
RequestImage	High	Texture	95,513	23,584,970	246.93	1,368	94	0.90%	2.68%
RebakeAvatarTextures	Low	Texture	33	1,809	54.82	67	54	0.00%	0.00%
ImagePacket	High	Texture	2,435,871	2,553,959,822	1,048.48	1,055	56	97.30%	290.45%
MapItemReply	Low	Land	139,515	142,500,720	1,021.40	1,419	102	89.94%	16.21%
LayerData	High	Land	13,440	7,835,310	582.98	1,371	54	4.95%	0.89%
MapBlockReply	Low	Land	11,992	5,120,609	427.00	1,278	87	3.23%	0.58%
Remainder	N/A	Land	33,374	2,989,341	89.57	130	82	1.89%	0.34%
LayerData	High	Wind	718,946	79,262,912	110.25	136	63	99.98%	9.01%
AgentCachedTextureResponse	Low	Wind	81	14,721	181.74	226	147	0.02%	0.00%
LayerData	High	Cloud	306,605	60,995,889	198.94	250	131	100.00%	6.94%
MapItemReply	Low	Resend	17,608	22,653,695	1,286.56	1,355	103	86.42%	2.58%
ImagePacket	High	Resend	2,570	2,692,838	1,047.80	1,055	136	10.27%	0.31%
MapBlockReply	Low	Resend	806	396,707	492.19	941	91	1.51%	0.05%
Remainder	N/A	Resend	4,108	471,476	114.77	1,232	40	1.80%	0.05%
TransferRequest	Low	Unknown	4,929	434,809	88.21	450	84	29.23%	0.05%
SetAlwaysRun	Low	Unknown	2,158	153,218	71.00	71	71	10.30%	0.02%
DirEventsReply	Low	Unknown	100	124,019	1,240.19	1,362	81	8.34%	0.01%
Remainder	N/A	Unknown	6,288	775,413	123.32	1,334	38	52.13%	0.09%

Table D.1: Information about packets.

#	Total			TCP			SL		
	Cons	**Data (KB)**	**Bandwidth**	**Cons**	**Data (KB)**	**Bandwidth**	**Cons**	**Data (KB)**	**Bandwidth**
1	823	169,795.143	168.761 Kbps	799	3,603.579	3.582 Kbps	24	166,191.563	165.180 Kbps
2	589	143,113.050	89.945 Kbps	580	2,762.896	1.736 Kbps	9	140,350.153	88.208 Kbps
3	966	195,753.760	128.321 Kbps	948	3,994.292	2.618 Kbps	18	191,759.468	125.703 Kbps
4	5	133.034	35.476 Kbps	5	133.034	35.476 Kbps	0	0.000	0.000 Kbps
5	675	185,729.472	119.912 Kbps	663	2,840.074	1.834 Kbps	12	182,889.397	118.079 Kbps
6	509	126,883.744	112.585 Kbps	501	2,231.763	1.980 Kbps	8	124,651.981	110.605 Kbps
7	856	170,604.090	134.268 Kbps	838	3,845.516	3.026 Kbps	18	166,758.574	131.241 Kbps
8	740	170,309.834	97.285 Kbps	731	3,669.391	2.096 Kbps	7	166,639.339	95.188 Kbps
9	760	208,917.688	118.484 Kbps	748	2,983.703	1.692 Kbps	12	205,933.984	116.792 Kbps
10	536	151,456.253	88.909 Kbps	533	2,047.178	1.202 Kbps	3	149,409.075	87.707 Kbps
11	1528	226,359.836	138.288 Kbps	1495	5,858.237	3.579 Kbps	33	220,501.599	134.709 Kbps
12	667	202,415.192	139.057 Kbps	660	2,858.749	1.964 Kbps	7	199,556.443	137.093 Kbps
13	597	154,112.648	132.599 Kbps	590	2,556.868	2.200 Kbps	7	151,555.780	130.399 Kbps
14	712	158,071.347	135.495 Kbps	705	3,124.243	2.678 Kbps	7	154,947.104	132.817 Kbps
15	628	170,648.325	88.995 Kbps	623	2,600.760	1.356 Kbps	5	168,047.565	87.639 Kbps
16	433	135,371.250	98.104 Kbps	430	1,612.236	1.168 Kbps	3	133,759.014	96.936 Kbps
17	537	163,540.885	143.520 Kbps	535	2,438.091	2.140 Kbps	2	161,102.794	141.380 Kbps
18	605	180,752.212	163.706 Kbps	600	2,647.983	2.398 Kbps	5	178,104.229	161.308 Kbps
19	993	115,381.062	81.005 Kbps	978	4,027.827	2.828 Kbps	15	111,353.234	78.177 Kbps
20	976	268,590.261	160.929 Kbps	957	3,985.261	2.388 Kbps	19	264,605.000	158.541 Kbps
21	579	129,745.220	93.611 Kbps	572	2,386.562	1.722 Kbps	7	127,358.657	91.889 Kbps
22	574	116,931.099	84.833 Kbps	567	2,519.616	1.828 Kbps	7	114,411.482	83.005 Kbps
23	696	172,192.948	124.970 Kbps	684	3,038.742	2.205 Kbps	12	169,154.206	122.765 Kbps
24	916	154,714.188	114.075 Kbps	902	3,725.222	2.747 Kbps	14	150,988.966	111.328 Kbps
25	553	111,381.877	96.906 Kbps	534	2,297.497	1.999 Kbps	19	109,084.380	94.908 Kbps
26	7	141.986	11.710 Kbps	7	141.986	11.710 Kbps	0	0.000	0.000 Kbps
27	493	131,869.955	103.988 Kbps	487	2,135.472	1.684 Kbps	6	129,734.483	102.304 Kbps
28	890	173,450.344	119.993 Kbps	883	3,776.654	2.613 Kbps	7	169,673.689	117.381 Kbps
29	813	173,196.626	116.719 Kbps	803	3,337.039	2.249 Kbps	10	169,859.587	114.470 Kbps
30	374	106,618.713	143.136 Kbps	368	1,863.390	2.502 Kbps	6	104,755.323	140.635 Kbps
31	914	179,135.565	144.435 Kbps	900	3,888.420	3.135 Kbps	14	175,247.146	141.300 Kbps
32	651	157,637.090	138.460 Kbps	642	3,953.028	3.472 Kbps	9	153,684.062	134.988 Kbps
33	592	243,563.759	164.556 Kbps	580	2,552.320	1.724 Kbps	7	240,952.784	162.792 Kbps
34	5	375.932	176.909 Kbps	5	375.932	176.909 Kbps	0	0.000	0.000 Kbps

Table D.2: The number of connections associated with each machine.

```
 1  router-c.cs.st-andrews.ac.uk (138.251.194.1)  0.609 ms  0.977 ms  1.224 ms
 2  oux-6513-jcb.st-and.ac.uk (138.251.2.201)  0.551 ms  0.543 ms  0.535 ms
 3  uos-gw-oux.st-and.ac.uk (138.251.2.49)  0.520 ms  0.514 ms  0.505 ms
 4  194.81.114.45 (194.81.114.45)  1.461 ms  1.452 ms  1.133 ms
 5  so-2-3-2.leed-sbr1.ja.net (146.97.42.49)  7.996 ms  7.985 ms  7.977 ms
 6  so-2-2-0.lond-sbr4.ja.net (146.97.33.30)  12.645 ms  12.614 ms  12.598 ms
 7  195.219.195.61 (195.219.195.61)  12.580 ms  12.560 ms  12.546 ms
 8  if-6-0.core1.AD1-Amsterdam.teleglobe.net (80.231.80.29)  26.358 ms  26.321 ms  26.298 ms
 9  Vlan83.icore1.AD1-Amsterdam.teleglobe.net (80.231.80.6)  30.158 ms  30.127 ms  29.285 ms
10  ge-6-16.car1.Amsterdam1.Level3.net (4.68.110.225)  22.540 ms  22.609 ms  22.618 ms
11  ae-31-53.ebr1.Amsterdam1.Level3.net (4.68.120.94)  32.593 ms  24.046 ms  24.006 ms
12  ae-1-100.ebr2.Amsterdam1.Level3.net (4.69.133.86)  38.791 ms  38.761 ms  38.746 ms
13  ae-2.ebr2.London1.Level3.net (4.69.132.133)  27.244 ms  23.796 ms  23.962 ms
14  * * *
15  ae-91-91.csw4.NewYork1.Level3.net (4.69.134.78)  96.221 ms ae-71-71.csw2.NewYork1.Level3.net (4.69.134.70)  95.235 ms ae-91-91.csw4.NewYork1.Level3.net (4.69.134.78)  96.443 ms
16  ae-83-83.ebr3.NewYork1.Level3.net (4.69.134.105)  102.593 ms ae-63-63.ebr3.NewYork1.Level3.net (4.69.134.97)  95.247 ms ae-73-73.ebr3.NewYork1.Level3.net (4.69.134.101)  96.029 ms
17  ae-6.ebr2.Dallas1.Level3.net (4.69.137.121)  132.010 ms  130.246 ms  131.339 ms
18  ae-62-62.csw1.Dallas1.Level3.net (4.69.136.138)  135.985 ms ae-82-82.csw3.Dallas1.Level3.net (4.69.136.146)  136.853 ms  137.551 ms
19  ae-1-69.edge3.Dallas1.Level3.net (4.68.19.8)  131.056 ms  133.059 ms  130.999 ms
20  LINDEN-RESE.edge3.Dallas1.Level3.net (4.78.224.14)  132.700 ms  132.677 ms  132.613 ms
```

Packet Type	Frequency	Throttle	Number	Bytes	Mean Size	Max Size	Min Size	% of Throttle	% of total
PacketAck	Fixed	NoThrottle	2,214	101,666	45.92	231	43	97.23%	3.09%
StartPingCheck	High	NoThrottle	68	2,760	40.59	45	40	2.64%	0.08%
AgentMovmentComplete	Low	NoThrottle	1	140	140.00	140	140	0.13%	0.00%
FetchInventoryReply	Low	Asset	4	4,168	1,042.00	1,208	610	81.17%	0.13%
FetchInventory	Low	Asset	1	967	967.00	967	967	18.83%	0.03%
AgentUpdate	High	Task	8,594	1,201,575	139.82	147	130	44.00%	36.56%
ObjectUpdateCompressed	High	Task	680	627,290	922.49	1,236	164	22.97%	19.09%
ObjectUpdate	High	Task	421	431,663	1,025.33	1,365	146	15.81%	13.14%
ViewerEffect	Medium	Task	976	145,010	148.58	192	96	5.31%	4.41%
ImprovedTerseObjectUpdate	High	Task	1,270	139,334	109.71	245	109	5.10%	4.24%
ObjectUpdateCached	High	Task	397	86,758	218.53	1,042	58	3.18%	2.64%
Remainder	N/A	Task	957	99,039	103.49	720	36	3.63%	3.01%
ImagePacket	High	Texture	316	330,727	1,046.60	1,055	66	96.36%	10.06%
ImageData	High	Texture	23	9,340	406.09	660	212	2.72%	0.28%
RequestImage	High	Texture	32	3,138	98.06	172	94	0.91%	0.10%
LayerData	High	Land	34	24,828	730.24	1,237	75	100.00%	0.76%
LayerData	High	Wind	339	34,803	102.66	109	97	100.00%	1.06%
LayerData	High	Cloud	145	28,419	195.99	215	179	100.00%	0.86%
CameraConstraint	High	Unknown	160	8,160	51.00	51	51	55.48%	0.25%
TestMessage	Low	Unknown	10	5,578	557.80	1,067	74	37.92%	0.17%
UUIDNameRequest	Low	Unknown	4	252	63.00	87	55	1.71%	0.01%
Remainder	N/A	Unknown	11	718	65.27	79	38	4.88%	0.02%

Table D.3: Information about packets.